12-2008

Agent-Organized Network Coalition Formation

Levi L. Barton
Utah State University

AGENT-ORGANIZED NETWORK COALITION FORMATION

by

Levi Barton

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

_____	_____
Dr. Vicki Allan Major Professor	Dr. Nicholas Flann Committee Member
_____	_____
Dr. Dan Watson Committee Member	Dr. Byron R. Burnham Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2008

ABSTRACT

Agent-Organized Network Coalition Formation

by

Levi Barton, Master of Science

Utah State University, 2008

Major Professor: Dr. Vicki Allan
Department: Computer Science

This thesis presents work based on modeling multi-agent coalition formation in an agent organized network. Agents choose which agents to connect with in the network. Tasks are periodically introduced into the network. Each task is defined by a set of skills that agents must fill. Agents form a coalition to complete a task by either joining an existing coalition a network neighbor belongs to, or by proposing a new coalition for a task no agents have proposed a coalition for. We introduce task patience and strategic task selection and show that they improve the number of successful coalitions agents form. We also introduce new methods of choosing agents to connect to in the network and compare the performance of these and existing methods.

(349 pages)

ACKNOWLEDGMENTS

I would like to thank my wife and son, my parents, and all of my instructors throughout my college career. I would especially like to thank Vicki Allan for all of the help and encouragement she gave throughout my time working with her. Without her insight, I would not have been able to complete this work.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Multi-agent coalition formation has been the topic of much research. Building on previous research [1], we study coalition formation among networked agents who seek to gather enough neighbors in the network to complete as many periodically created tasks as possible. We use performance as a metric to compare different methods presented. We define *performance* as the number of tasks completed divided by the total number of tasks introduced to the network. We examine the effects on performance of different algorithms for choosing tasks and neighbors in an agent-organized network.

1.1 Agent-Organized Network

An agent-organized network is defined as a set of agents who are connected by edges in a bidirectional graph. Tasks are visible to all agents, and tasks are accomplished by forming teams among agents who make up a connected component in the network graph. The choice of whom to be connected to in the network becomes very important when considering future coalitions that may be formed. Agents do this by using various strategies for managing their network connections to other agents, which we call *rewiring*.

1.2 Rewiring Strategies

We use two existing rewiring strategies (structural and performance rewiring) [1], and add several others (diversity, egalitarian, and inventory rewiring). Agents who use structural rewiring seek to connect to agents who have the most network connections. Structural rewiring results in a network wherein a small group of agents (hub nodes) have

many network connections, and the majority of the agents (spoke nodes) have few network connections. Agents who use performance rewiring seek to connect to agents who have been the most successful at completing coalitions in the past. Agents who use diversity rewiring seek to connect to agents who have a skill not possessed by the agent who is doing the rewiring. Agents who use egalitarian rewiring seek to connect to agents who have the fewest network connections. Agents who use inventory rewiring seek to connect to agents who have a skill that is not possessed by agents in the existing network neighborhood, or to agents who possess a skill that has been left unfilled by failed tasks.

1.3 Task Selection and Rejection

We look at choices agents have when committing to partially formed coalitions (coalitions that still have not had all required skills filled). First, we look at two options for choosing which partial coalition to commit to: basic and strategic task selection. Basic task selection chooses the first coalition with an unfilled skill that the agent possesses, while strategic task selection chooses the coalition with an unfilled skill the agent possesses that is the closest to becoming a complete coalition. Second, we look at agents who have made a commitment to a partial coalition, with agents choosing to be either task patient or task impatient. Task patient agents remain committed to the partial coalition until the coalition has formed, or the task the coalition is working on has expired. Task impatient agents revoke their commitment to a partially formed coalition if the number of unfilled skills in the task the coalition is seeking to work on is greater than the number of uncommitted agents the agent is connected to in the network.

We also examine the effect changing the rate of tasks added to the network has on the percentage of tasks completed. We look at the effects of different rates of task

creation on the performance of different agent types, discovering which agent strategies best deal with high task loading.

1.4 Information Sharing

We analyze including more information when rewiring by gathering additional information from agents within a specified depth of the inquiring agent. We define the *communication depth* to be the maximum length in network links from the rewiring agent to the agents used to gather information. We compare the performance of each agent type at different depths of communication.

1.5 Skill-Supply and Skill-Demand

We test the effects of changing the skill-supply and skill-demand. We call the skills possessed by the agents in the network the *skill-supply* and the skills required by the tasks created the *skill-demand*. The skill-supply and skill-demand are made to conform to three skill distributions: uniform, overlapping, and congregate. Each skill distribution gives each skill a varying probability of being represented in a task or possessed by an agent in the network. We examine the performance of the different agent types under various combinations of skill-supply and skill-demand.

1.6 Contribution

We show the benefit of using strategic task selection and task impatience to increase the performance of existing rewiring strategies. We show that adding strategic task selection and impatient task selection also improve performance when there is a high task load.

We also demonstrate placing an upper bound on the number of network connections an agent can have in the agent-organized network does not adversely impact overall performance as long as the upper bound is not too small. This minimizes the risk that one agent failing cripples the performance of the network.

Our work adds new rewiring strategies to those previously employed in an agent-organized network. The best new rewiring strategy we introduce is inventory rewiring. This is shown in the performance improvement inventory agents provide when skill-demand is non-uniform. The existing structural rewiring method works best when uniform skill-demand exists. However, uniform skill-demand would likely be rare in real-world applications of agent-organized networks, making inventory agents the best choice.

1.7 Real-World Application

Agent-organized networks are well suited to dynamic environments wherein tasks require flexibility due to changing resource requirements and incomplete information. An example of this is a group of agents tasked with fighting a wildfire. The tasks the agents need to complete are constantly changing as the fire burns more acreage. The agents may not have complete information about the entire environment surrounding the fire. An agent-organized network would be able to provide a flexible framework for agents to deal with these dynamic requirements.

1.8 Paper Layout

This thesis contains three papers submitted to various conferences. They represent three separate sets of results focusing on three different areas of emphasis, all relating to our work on coalition formation in an agent-organized network.

Paper 1 (Chapter 2 – Methods for Coalition Formation in Adaptation-Based Social Networks, accepted by CIA '07) introduces our work on agents who are task impatient, and agents who choose tasks strategically. We also examine the effects of limiting the maximum number of network connections an agent can have, as well as the rate of introducing new tasks to the agents in the network.

Paper 2 (Chapter 3 – Information Sharing in an Agent-Organized Network, accepted by IAT '07) introduces agents that use inventory and egalitarian rewiring. We examine the effects on agent performance of increasing the amount of information available to agents when rewiring for structural, performance, egalitarian, and inventory agents.

Paper 3 (Chapter 4 – Adapting to Changing Resource Requirements for Coalition Formation in Self-Organized Social Networks, accepted by IAT '08) explores how agents perform with different congregations of skill-supply (the skills agents possess) and skill-demand (the skills required to complete a task). Congregate skill congregation (three groups of skills are used to produce skills agents possess and skills required in tasks), overlapping skill congregation (three groups of skills that overlap the neighboring congregations), and uniform skill congregation (one group containing all skills possible is used when creating skills agents possess and skills required in tasks) are examined.

References

[1] M. Klusch and A. Gerber. Dynamic coalition formation among rational agents. *IEEE Intell. Syst.,* 17: 42–47, 2002.

CHAPTER 2

METHODS FOR COALITION FORMATION IN ADAPTATION-BASED SOCIAL NETWORKS[1]

Abstract

Coalition formation in social networks consisting of a graph of interdependent agents allows many choices of tasks to select and with whom to partner in the social network. Nodes represent agents, and arcs represent communication paths for requesting team formation. Teams are formed in which each agent must be connected to another agent in the team by an arc. Agents discover effective network structures by adaptation. Agents also use several strategies for selecting the task and determining when to abandon an incomplete coalition. Coalitions are finalized in one-on-one negotiation, building a working coalition incrementally.

2.1. Introduction

Much of the work in the area of coalition formation has focused on forming optimal, stable coalitions. These methods require complete information and require substantial computational time [2, 7]. In a dynamic environment, it may not be possible for an agent to maintain this complete view due to inaccurate information provided by faulty sensors or unreliable communications [10]. Our environment is different from many others in that teams must form a connected component in the social network. Most coalition formation problems are specified to highlight the problems in deciding which

[1] Coauthored by Levi Barton and Vicki Allan.

agents should work together to form teams. Our version of the problem adds the additional constraint of the structure of the agents forming a coalition.

Coalition formation among agents in dynamic environments presents additional challenges compared to coalition formation in static environments. The use of traditional coalition formation methods that compute a kernel to determine a coalition's division of utility is NP-Hard [6] and centralized. Both the changing nature of the tasks in a dynamic environment and the restrictions of teammates imposed by the social network render traditional coalition formation methods unusable.

Dynamic events in the environment change the nature of coalition formation. Tasks enter the system over time. The utility of a task may decrease over time. We study both the self-interested agent algorithm and how the structure of the graph affects performance. Because the correct structure cannot be known a priori, we allow agents to adapt the network structure (sometimes termed *rewiring*) in order to determine the desirability of certain structures. Agent capabilities include joining an existing team, initiating a new team, waiting, or rewiring the connections. If team formation is successful, participating agents become active in the task, complete the task, and then rejoin the set of available agents.

In previous work, sub-optimal coalitions have been studied in coalition formation with incomplete information and with limited computational resources [9, 10]. In an effort to reduce computational expense, agents can prune the list of possible coalition partners to those considered most trustworthy or beneficial. These groups of agents who come together based on trust developed over past interactions are called *congregations* or clans [5]. By considering fewer potential coalition partners, as in our model, agents using

clans can more easily compute possible coalitions within the time constraints imposed by the environment with the value of coalitions formed being good enough, or *satisficing* [5, 10].

This research implements a multi-agent system that uses an agent-organized network to facilitate coalition formation. The agents communicate tasks to neighboring agents to initiate the formation of a coalition to perform the task. When deciding which agents to include in the coalition, the agent forming the coalition considers the agent's skills.

2.2. Agent-Organized Networks

An *Agent-organized network (AON)* is a set of inter-connected agents who collectively manage the structure of this network of agents by making individual decisions about which agent to connect to based on local information [3]. In our environment, *nodes* represent agents, and *links* represent social structure. The *neighbors* of an agent are all agents connected to it via a network arc. The arcs may represent a variety of physical constraints, such as physical distance, limited communication, trust, or organizational hierarchy. Teams must be connected components in the social network.

The initial set of network arcs are established by connecting each agent to any agent within distance d of its location. For our tests, we used d=100 in order to compare with other authors' work [3].

Tasks are introduced to all agents within a region of interest, and are executed by agents who are connected in the network. A task is composed of a set of subtasks requiring specific skills. In our system, the total number of possible skills is denoted *SkillMax*. A task is specified by an array skillCt[1.. *SkillMax*] in which skillCt[i] indicates

the number of agents possessing skill i that are required by the task. Thus, in an environment with five possible skills, a task with skillCt = [1,2,0,0,3] requires six agents: one with skill 1, two with skill 2, and three with skill 5. In our system, skills are generated with a uniform distribution. We define the number of skills per task as *TaskSkills*, and the number of possible agent skills as *AgentSkills* (with *AgentSkills*=1 for our tests).

In order to compare our results directly with those of [3], we performed identical tests. System parameters are summarized in Table 2.1. The length of the interval between task generations is *GenInterval*. If *GenInterval* = 1, one task is generated every iteration. If *GenInterval* = 5, one task is generated every five iterations. For simplicity, the number of skills required per task is constant, and all tasks have the same utility. Tasks are announced for a time interval *AnnouncePeriod*, which depends on the number of skills

Table 2.1. System parameters

System Parameter	**Meaning**	**Default Value**
SkillMax	Number of different skills in system	10
TaskSkills	Number of skills per task	10
AgentSkills	Number of skills per agent	1
GenInterval	Iterations between task generation	10
AnnouncePeriod	Number of iterations task is schedulable	10
CommitTime	Maximum time an agent will stay in a partially formed team	10
TaskDuration	Iterations required to complete task	10
DegreeLimit	Maximum degree of any node in system	20
RewireFrequency	Probability agent rewires in a given turn	1/N (N=# agents)

required. If the team for a task has not formed within AnnouncePeriod iterations, the task is removed from the system and counted as a failed task. If a team fails to form during a time interval of length *CommitTime*, an agent abandons the team. If all members abandon the team, the task returns to the available list (if its AnnouncePeriod has not expired). The duration of each task is specified as its *TaskDuration*.

In forming teams, an agent may only communicate with its neighbors. Once a neighbor joins the team for a task, its neighbors are allowed to join the team. We term the neighbors of a neighbor, *FOAF* (friend of a friend) [1]. Each agent has a single skill, a position on the grid, and a neighbor set. Agents are in one of three states: committed, active, or uncommitted, as shown in Figure 2.1. A *committed* agent has joined a partial team. An *active* agent has joined a team that is now fully formed and is currently involved in completing the associated task for TaskDuration. An *uncommitted* agent is not associated with a team.

Tasks require a team consisting of agents with a specific set of skills. A task consists of a taskID, an associated team (if any), an AnnouncePeriod, and a TaskDuration. A team consists of a taskID and a set of agents. A *complete team* has a match between required skills for the task and agents of the team. A *partial team* requires additional agents, as some skill requirements are not met. Tasks are announced to agents within a fixed radius of the event. To eliminate variability in the results, in these tests, all tasks are known by all agents.

Arcs are initially assigned between nodes that are closer than a fixed parameter, but are changed during the process of rewiring. The degree of a node is limited to control

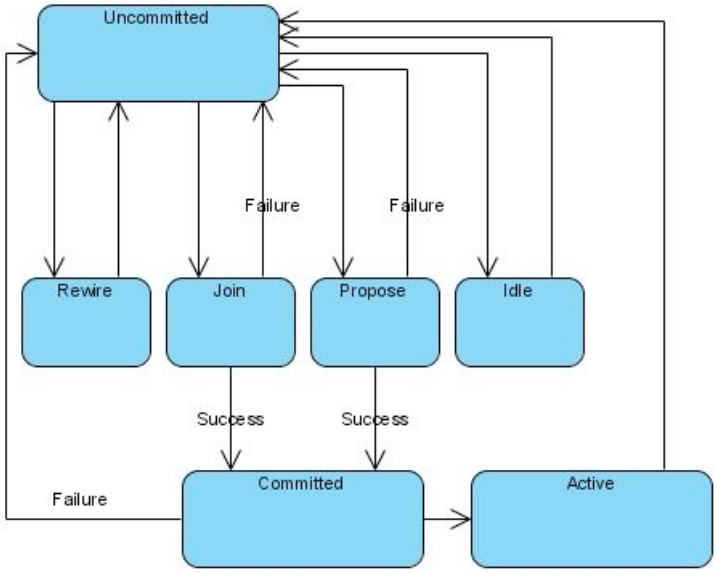

Figure 2.1. AON agent state diagram

communication. An existing partial team can only be joined by an agent if one of its neighbors has already joined the team.

The set of all uncommitted agents that are within d links of agent i and are connected to i via a chain of uncommitted agents is called a *circle of radius d*. For our tests, d is 1. Agents know the skills of the agents within their circle, termed the *circle skills*. A skill match is a measure of how closely the remaining skills for a task can be satisfied by the circle skills. *Skill match ratio* is computed as the ratio of the needed skills that can be provided by the circle divided by the total number of remaining skills to be acquired. The view of agents in the circle is limited by its radius; therefore, a skill match of less than one does not necessarily mean the team cannot be formed. Since the availability of an agent changes over time, the skill match is a rough estimate of the ability of the team to successfully form. As we increase the circle radius, we would hope

to achieve a skill match of 1. *Needed agents* is the total number of agents that are missing from the partial team.

At each iteration, an uncommitted agent can remain idle, join an existing partial team, rewire, or initiate a team for a task that is a member of its *EligibleTasks* set. The definition of *best* varies depending on agent type.

Selecting the best team to join: An agent selects an eligible task (if any) from tasks joined by its set of committed neighbors.

Selecting the best task to initiate: Rather than joining an existing team, an agent may choose to start a new team. A task is *viable* for an agent if its AnnouncePeriod has not expired, it requires a skill the agent possesses, and the task is not completed. A task is termed *unclaimed* if no partial or full team exists that is associated with the viable task. Only unclaimed tasks are considered for initiation.

Remaining idle: An agent may decide to remain idle as a strategy to remain available for tasks that do not yet involve one of the agent's neighbors or future proposed tasks.

Rewiring: The agent type controls adaptation method, which determines the conditions for rewiring, the frequency of rewiring, and the target of rewiring.

2.3. Agent Types

An agent type consists of three features: tasks selection method, task patience, and adaptation approach.

Task selection: There are two methods of picking activities*: basic* and *strategi*c. Basic agents first decide if they want to rewire. If they do not rewire, they then consider the tasks, in order of task generation. This gives preference to proposing/joining older

tasks. If a neighbor has joined the team for a task, the agent joins the team (if it has a necessary skill). If a viable task is unclaimed, an agent will propose the task with probability proportional to the number of uncommitted neighbors. There is no clear choice between initiating a team and joining an existing team, but since older tasks are given preference and there are more joiners than proposers needed for a coalition, joining is more likely.

Strategic agents first consider the set of tasks claimed by neighbors. The task with the greatest ratio of committed agents (assigned skills/total skills is highest) is selected. This ratio is called the *committed ratio*. If the committed ratio is greater than some agent-specified threshold, the team for the task is joined. Thus, strategic agents give preference to joining an existing task over rewiring, proposing, or waiting. The agent rewires based on the *rewiring frequency*. Strategic agents choose to wait based on the desirable waiting probability (which is normally computed from the total number of skills required for a task). Agents that do not choose to rewire, wait, or join will propose the task for which the skill match ratio is the best as long as it exceeds some agent-specified threshold (.34 in our tests).

Task patience: CommitTime is equal to AnnouncePeriod in patient agents. In impatient agents, an agent may abandon a task if certain success indicators are not present. For our tests, the task is abandoned when the count of uncommitted neighbors is less than the number of skills needed.

*Adaptation approach***:** An agent has the ability to remove a link to a neighbor and create a link to a new target. This rewiring is intended to increase the performance of the system. We define system performance as the fraction of tasks completed. We define

node performance as the fraction of successful teams joined over the number of teams attempted by a node. We define *system efficiency* as the fraction of time agents spend actively executing tasks. Agents executing tasks are termed *working*.

Rewiring involves an agent selecting one of its adjacent edges to disconnect from its current target and reconnect to another target. The edge is jointly owned by source and target, so either agent can initiate the rewiring, and no agent can refuse to be connected to a particular node (unless its maximum degree has been exceeded). We study three types of rewiring approaches: *structural, performance,* and *diversity*. Structural rewiring is motivated by findings in network topology [11]. Defined in [3], performance rewiring is based on the desire to be connected to agents that have higher performance. Diversity rewiring is based on the desire to be connected to an agent with a different skill than the skill possessed by the rewiring agent.

In performance adaptation, the adaptation trigger is based on local performance. Local performance is only valid when the number of teams joined is larger than a set number (five in our case). Local performance is computed as the number of successful teams joined divided by the number of teams joined. When an agent's local performance is less than the average of its neighbors' performance, rewiring is triggered: the agent removes its link to the worst performing neighbor and replaces it with a link to the best performing neighbor of its best performing neighbor.

In structural adaptation, rewiring is independent of performance and occurs with probability 1/N where N is the number of agents. An agent randomly picks a link and replaces it with a link to a FOAF based on preferential attachment. *Preferential*

attachment means that the probability of a particular target is its degree divided by the total degree of all the neighbor's neighbors.

In diversity adaptation, rewiring is independent of performance and occurs with probability 1/N where N is the number of agents as in structural rewiring, but the only requirement for selecting a target is that the target must have a different skill than the agent.

If we limit the degree of a node in the graph, we say the adaptation is *bounded*. The idea behind bounded rewiring is to eliminate the communication bottleneck. On network creation, arcs to a node of degree DegreeLimit are ignored. During adaptation, nodes with degree DegreeLimit are invisible to nodes seeking a new target.

2.4. Results

We conducted a variety of tests to evaluate both the activity selection and the adaptation approaches. To compare the methods, we repeated each test 50 times with 100 nodes and computed 95% confidence intervals. Each test executed for 2000 cycles, with adaptation beginning at time step 1000. Other parameters were varied as specified. Table 2.2 shows the task selection, task patience, and adaptation used by each agent type.

Table 2.2. Agent types

Agent Type	Task Selection	Task Patience	Adaptation Approaches
Performance	Basic	Patient	Performance
Structural	Basic	Patient	Structural
Structural Strategic	Strategic	Patient	Structural
Structural Strategic Impatient	Strategic	Impatient	Structural
Diversity Impatient	Basic	Impatient	Diversity
Diversity Strategic Impatient	Strategic	Impatient	Diversity

Since the total number of arcs in all methods is unchanged by rewiring, structural rewiring creates some nodes of high degree (hubs) connected to others of small degree (spokes). Thus, equal degree nodes are sacrificed to achieve high degree nodes.

In Figure 2.2, adaptation is disabled showing the differences in performance resulting from agent decisions in task selection and task patience. Performance and structural agents use basic task selection and are patient, resulting in the lowest performance at 0.24. The structural-strategic agent uses strategic task selection and is patient, resulting in a performance of 0.29, with a 5% improvement over the basic patient agents. The structural-strategic-impatient agent uses strategic task selection and is impatient, resulting in a performance of 0.36, with a 7% improvement over the structural strategic agent. Thus, task impatience and strategic task selection each individually provide improvements in performance. They also complement each other, offering even better performance when used together.

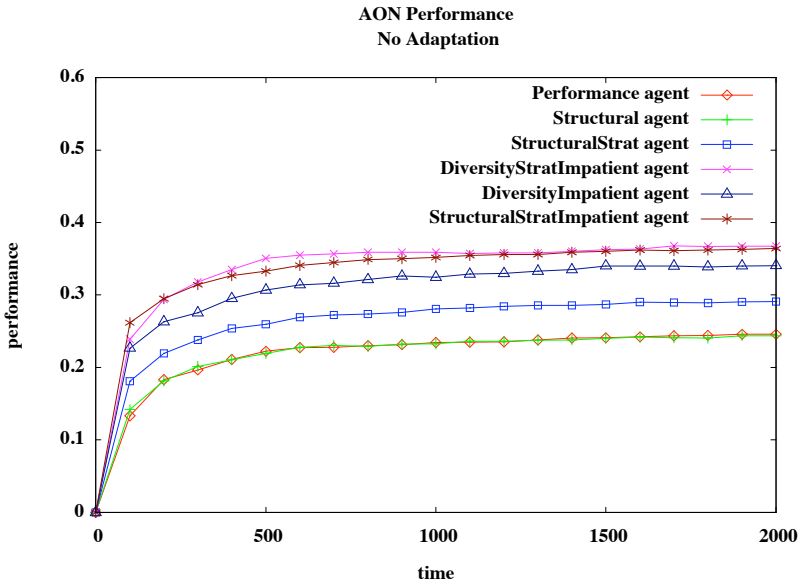

Figure 2.2. AON performance with no adaptation

Figure 2.3 shows the effect on performance when a degree limit is imposed on performance agents in an AON network. When the degree limit is 5, the performance is much lower, even before adaptation, as agents fail to have a network structure that allows for adequate social interaction to form teams as quickly, resulting in more failed teams. By increasing the degree limit to 10, the performance increases from 0.13 to 0.24, nearly doubling the performance. This reflects the benefits of having better connectivity, resulting in a higher probability of a successful team. Diversity agents exhibit a similar pattern of response to degree limits.

Figure 2.4 displays the effects of imposing a degree limit on structural agents in an AON network. Once again, when the degree limit is 5, the performance is relatively low and does not improve with rewiring. When the degree limit is increased to 10, the performance before adaptation begins (at time step 1000) also increases from 0.13 to

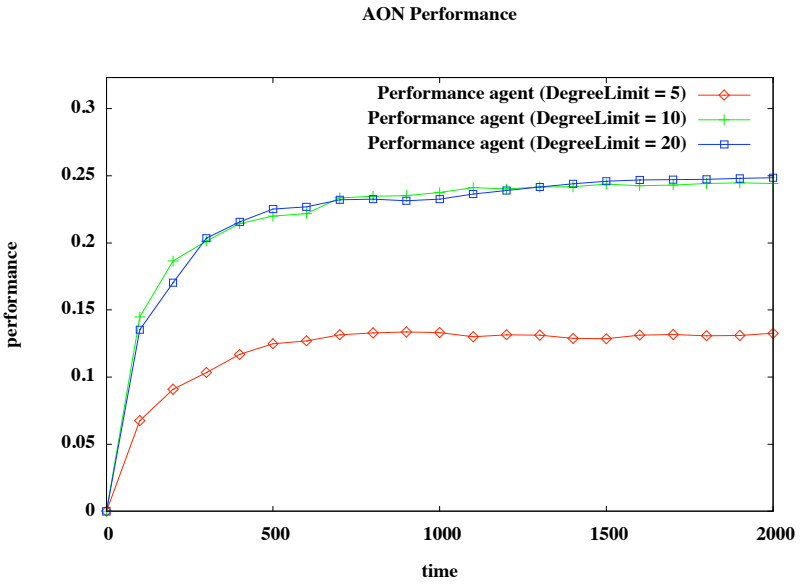

Figure 2.3. Limiting maximum agent degree on performance agents

0.24. After adaptation, the performance increases due to the hub network structure resulting from structural adaptation. When the degree limit is increased to 20, the performance before adaptation begins is 0.23, which is nearly the same performance level using a degree limit of 10. After adaptation, the performance increases at a greater rate and achieves a higher level than when the degree limit is 10.

The performance of the various methods is sensitive to the specific paramenters. In one test, each agent is only committed to the task for 4 iterations (CommitTime=4). The task is advertised for 10 iterations (AnnouncePeriod=10), so it is possible that a task will succeed even if the initial partial team abandons it. There are eight skills possible (SkillMax=8) and five skills per task (TaskSkillCount=5). Two tasks are introduced per time step (GenInterval=0.5). Structural-strategic-impatient and diversity-strategic-impatient agents out-perform the other agents. Structural-strategic and diversity-impatient agents perform nearly equally due to a small maximum committed time being enforced

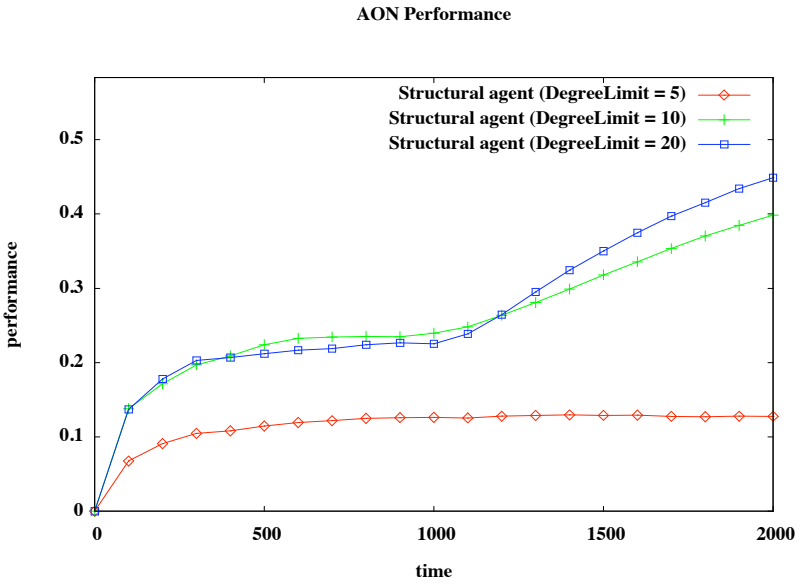

Figure 2.4. Limiting maximum agent degree on structural agents

for all methods. The advantage usually enjoyed by the strategic agent is decreased as all methods allow a task less time to succeed before attempting to move to other tasks.

When we consider only the improvement occurring because of adaptation, the agent types rank differently with respect to relative improvement than with respect to performance. This indicates that agents with poor performance have more to gain from adaptation while those with better performance are unable to achieve the same level of improvement.

Once a team for a task is joined, the agents that are task patient (performance, structural, and structural-strategic agents) stay with the task until it has expired (as CommitTime equals AnnouncePeriod). An impatient agent has a significant advantage (Figure 2.5) because it has the choice of abandoning an unpromising task before the task expires, which allows other agents to complete the task and frees the agent to pursue

more profitable activities. Structural-strategic is still competitive as it is more selective in the initial choice of a task.

Consider the set of test cases (Figure 2.6 and Figure 2.7) that differ only in the degree limit. Agents are committed for a maximum of 10 iterations to a task (CommitTime=10). The tasks are advertised for 10 iterations (AnnouncePeriod=10).

In Figure 2.6, the node degree is limited to 10, allowing an increase of performance after adaptation is allowed at time step 1000. Strategic and impatient methods have a higher initial level of performance before adaptation is allowed, but methods using structural adaptation show more improvement in performance than other adaptation methods. The structural-strategic-impatient agent has the benefit of both a high initial performance level, a result of the combination of strategic task selection and

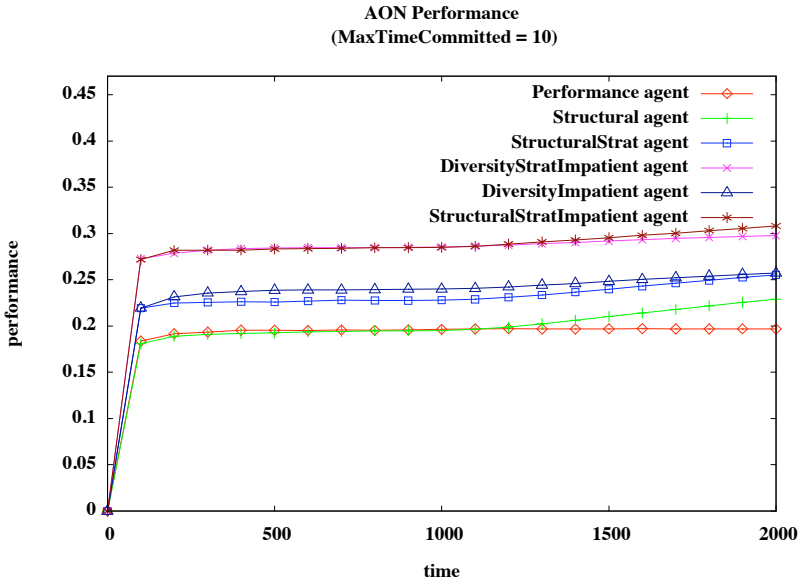

Figure 2.5. AON performance with maximum time committed of 10 iterations

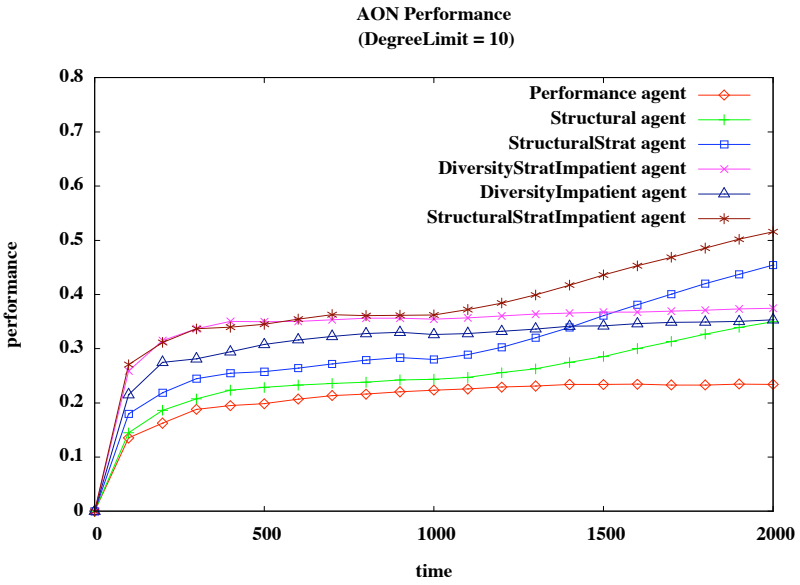

Figure 2.6. Performance with degree limit of 10

task impatience. The structural-strategic-impatient agent also shows significant improvement in performance with adaptation, resulting from the structural rewiring strategy, making it the best performing method of all. We made significant improvements over the structural results of [3] by changing the task selection algorithm and the team commitment. While strategic task selection requires increased knowledge, the team commitment limitation requires no special intelligence. Note that while structural-strategic-impatient is smarter, it increases in performance at the same rate as the less effective structural agent due to their shared use of structural rewiring.

In Figure 2.7, the degree limit is increased to 20. As expected, the performance of some agent types does not increase when adaptation is begun at time step 1000 as shown by the flat curves. Increasing the degree limit to 20, the performance levels (after adaptation) increase, especially for the structural rewiring methods. Structural rewiring methods are able to achieve nodes of a high degree, increasing the probability of being

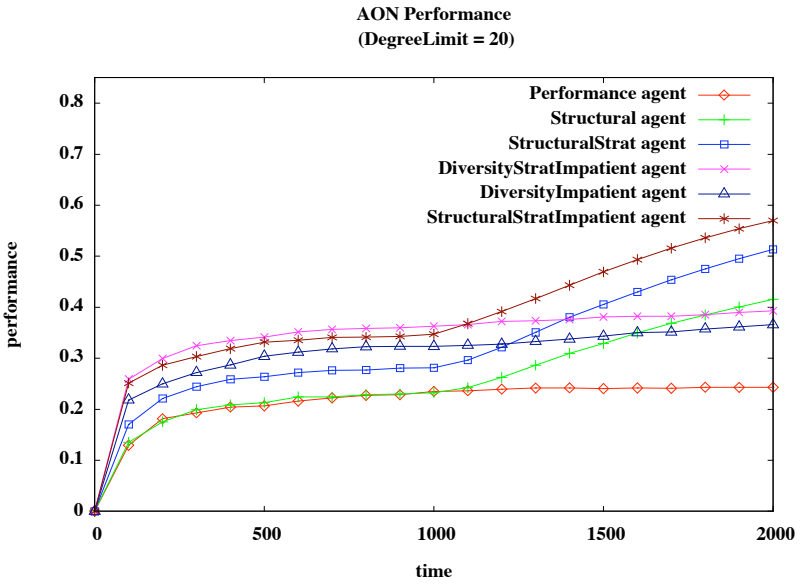

Figure 2.7. Performance with degree limit of 20

able to form a successful team, given the number of neighbors who possess the skills required.

Each time the degree limit is increased, the structural rewiring methods experience an increase in the highest degree of a node in the network. The other rewiring methods seek for neighbors with either greater performance or a more diverse skill set. Therefore, they do not experience an increase in the highest node degree.

As the degree limit is increased to 30 and above, the performance increases achieved due to the higher degree limit decreases. This can be seen in Figure 2.8, where the final performance level achieved by performance, structural, and diversity agents are shown for degree limits of 5 through 100. When an agent has a sufficient number of

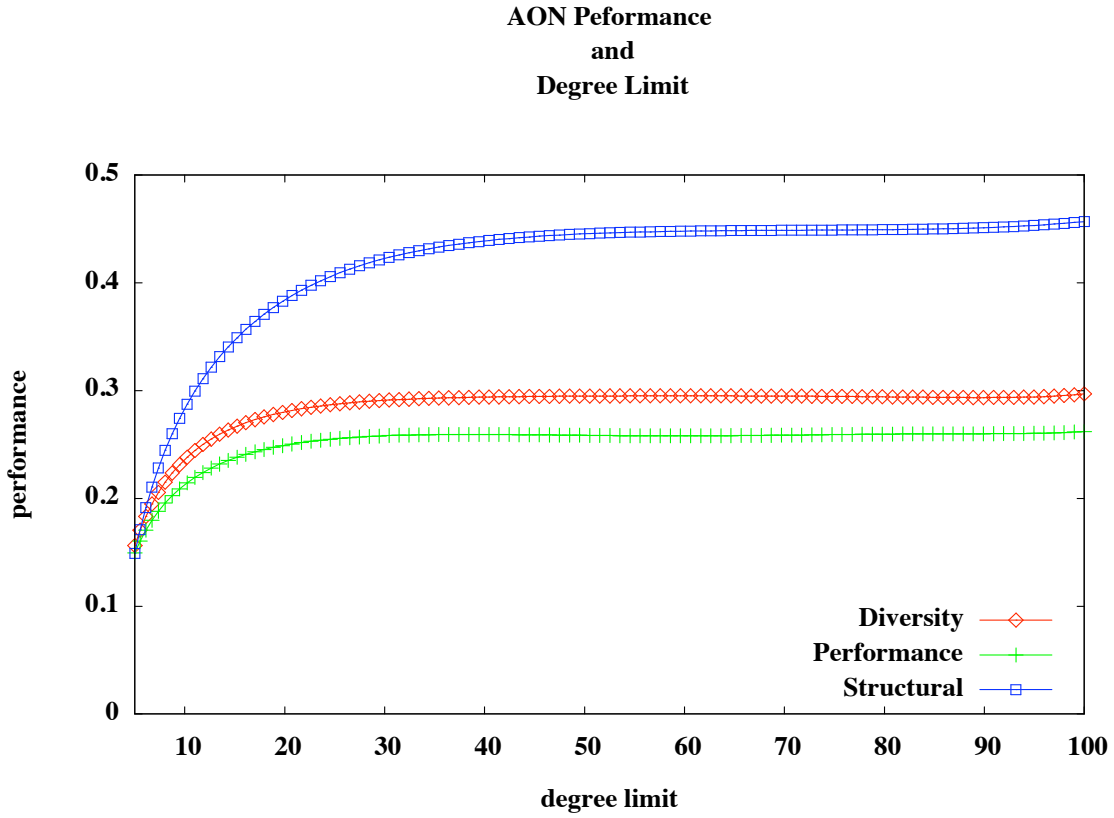

Figure 2.8. AON performance vs. degree limit

neighbors, it will not continue to benefit from a higher degree due to the number of skills per task (10 in these tests). For example, if an agent has 20 neighbors, and needs to fill a task with 10 skills, it will never need to use more than half of its neighbors to successfully complete this task. Also, while completing a task with a successfully formed team, the neighbors not included in the team may be unable to form an additional team successfully because of the resulting shortage of potential coalition partners when the hub node connecting them is not available. This puts a performance limit on forming networks in which high degree nodes exist.

We also experiment with the proportion of tasks to agents. While holding the agent count constant, we increase the number of tasks introduced at each time step. In

these tests, there are 100 agents. Tasks need 10 skills, and there are 10 possible skills. Since tasks have duration 10 and task formation takes an average of three iterations, the system of agents cannot possibly complete more than 1.3 tasks per iteration. Thus, introducing two tasks per iteration gives the agents many tasks for selection (Figure 2.9).

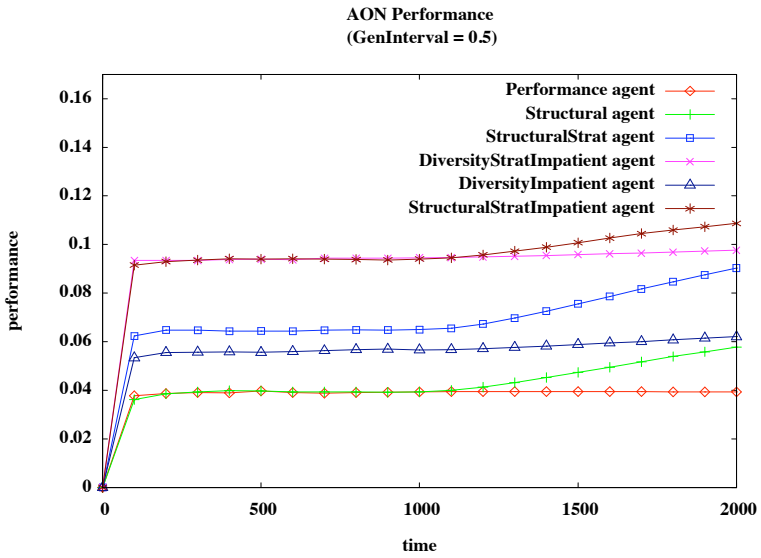

Figure 2.9. Performance with two tasks per iteration

Agents who use strategic task selection do very well, as they can pick the best tasks. As before, agents using structural adaptation also do well, as the resulting structure means that the skills are available to complete many tasks. Performance agents perform poorly, likely because clusters of under-performing agents are not motivated to rewire because they *are* doing as well as their neighbors.

Notice, however, that the performance is less than 0.12 (compared to earlier tests showing performance levels of nearly 0.6). The performance necessarily decreases, as it is not possible to complete all the tasks and performance measures percentage of tasks completed. However, the performance is even lower than one might guess. This can be

attributed to the fact that many competing tasks seduce more agents to join a team for a task, resulting in many partially formed teams with no waiting agents to complete them.

2.5. Previous Work

Gaston and des Jardins [3] implement multiagent team formation in which tasks are globally known. Gaston and des Jardins rewire based on performance and structural metrics. Under performance adaptation, an agent whose performance is less than the performance of its neighbors is allowed to reconnect to the best of the neighbor's neighbors. Under structural adaptation, agents rewire to select a target from the set of neighbor's neighbors. The probability of connecting to an agent is proportional to a node's degree. Our results show the performance of Gaston and des Jardins' structural agents can be improved upon by making strategic task selections and by using task impatience.

Soh and Tsatsoulis [9, 10] present their method of creating suboptimal coalitions in a dynamic environment to deal with partial information and meet time constraints imposed by the real-time nature of the environment. They apply their method in a distributed target-tracking environment wherein the agents are stationary. Our method is based on their work, with an added challenge of forming coalitions among agents who navigate within a dynamic environment.

Griffiths and Luck [4] discuss their method of coalition formation for clans whose cooperation is of medium-term duration. They combine agent motivations and agent trust to determine when to create and terminate these clans. They introduce a kinship motivation, wherein agents determine the probability of encountering tasks that require cooperation based on tasks recently encountered.

Soh and Chen [8] define a collaboration utility between an agent-based history of successful collaboration and measures the reliance of one agent on another in forming coalitions. While no discrete labeling of clan is used, the collaboration utility specifies a loose clan in which trusted agents are more likely to be recipients of proposals and are more likely to be accepted.

2.6. Conclusion

We have introduced strategic task selection, task impatience, and diversity adaptation. We have shown the benefits of using these agent strategies in an AON network. Agent performance is improved when using strategic task selection versus basic task selection. Agent performance is also improved when using task impatience versus task patience. The best agent performance has been shown to result from using strategic task selection, task impatience, and strategic rewiring, the rewiring method that provides the greatest increase in performance in AON networks. Future research should examine the possibility of using a variety of agent strategies within the same network to improve performance.

Strategic agents have been shown to perform well under high task loads due to their ability to choose the task most likely to succeed. Task impatient agents have been shown to improve agent performance, but are affected more by high task loads.

AON networks containing nodes of high degree show increased levels of performance. However, high degree nodes also incur higher communication costs. We have shown that moderate maximum degree limitations do not significantly affect AON performance. This allows implementations of real-world AON applications to impose

maximum degree limitations to decrease communication costs without experiencing degraded performance.

References

[1] D. Brickley and L. Miller. (Nov, 2007) FOAF: The 'friend of a friend' vocabulary. Available online at http://xmlns. com/foaf/0.1/.

[2] V. Dang and N. Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *AAMAS*, 2004, 564–571.

[3] M. E. Gaston and M. des Jardins. Agent-organized networks for dynamic team formation. In *AAMAS*, 2005, 230–237.

[4] N. Griffiths and M. Luck. Coalition formation through motivation and trust. In *AAMAS*, 2003, 17–24.

[5] B. Horling and V. Lesser. A survey of multi-agent organizational paradigms. *Knowledge Engineering Review,* 19:281–316, 2005.

[6] H. C. Lau and L. Zhang. Task allocation via multi-agent coalition formation: Taxonomy, algorithms and complexity. In *15th ICTAI*, 2003, 346–350.

[7] T. Sandholm and V. Lesser. Coalitions among computationally bounded agents. *Artificial Intell., Special Issue on Economic Principles of Multi-Agent Systems,* 94:99–137, Jan. 1997.

[8] L.-K. Soh and C. Chen. Balancing ontological and operational factors in refining multiagent neighborhoods. In *AAMAS*, 2005, 745–752.

[9] L.-K. Soh and C. Tsatsoulis. Reflective negotiating agents for real-time multisensor target tracking. In *IJCAI*, 2001, 1121–1127.

[10] L.-K. Soh and C. Tsatsoulis. Utility-based multiagent coalition formation with incomplete information and time constraints. In *IEEE International Conference SMC,* vol. 2, 2003, 1481–1486.

[11] H. P. Thadakamalla, U. N. Raghaven, S. Kumara, and R. Albert. Survivability of multiagent-based supply networks: A topological perspective. *IEEE Intell. Syst.,* 19:24-31, 2004.

CHAPTER 3

INFORMATION SHARING IN AN AGENT-ORGANIZED NETWORK[2]

Abstract

Coalition formation in social networks consisting of a graph of interdependent agents allows many choices of tasks to select and with whom to partner in the social network. Nodes represent agents, and arcs represent communication paths for requesting team formation. Teams are formed in which each agent must be connected to another agent in the team by an arc. Agents communicate with agents within n network links in their surrounding network. These agents are considered part of an agent's local neighborhood. We introduce agents who maintain a database of skills possessed by agents in their local neighborhood, using the information gathered to discover effective network structures. This method is compared to structural agents, who seek to connect to agents who have a large number of network connections and egalitarian agents, who seek equal distribution of connections among agents, resulting in a dense network structure.

3.1. Introduction

Much work has been done in the area of coalition formation that seeks to form optimal, stable coalitions. The methods used to form these optimal coalitions require complete information and substantial computational time [3, 8]. When the agent's environment is dynamic, it may not be possible for an agent to have a complete view of all agents, due to inaccurate information provided by faulty sensors or unreliable communications [10]. Our environment is different from many others in that teams must form a connected component in the social network. Most coalition formation problems

[2] Coauthored by Levi Barton and Vicki Allan.

are specified to highlight the problems in deciding which agents should work together to form teams. Our version of the problem adds the additional constraint of the structure of the agents that form a coalition. We look at how much information agents share with other agents in the network in an effort to overcome an incomplete view of any single agent.

Coalition formation among agents in dynamic environments presents additional challenges compared to coalition formation in static environments. The use of traditional coalition formation methods that compute a kernel to determine a coalition's division of utility is NP-Hard [7] and centralized. Both the changing nature of the tasks in a dynamic environment and the restrictions of teammates imposed by the social network render traditional coalition formation methods unusable.

Dynamic events in the environment change the nature of coalition formation. Tasks enter the system over time. The utility of a task may decrease over time. We study both the self-interested agent algorithm and how the structure of the graph affects performance. Because the correct structure cannot be known a priori, we allow agents to adapt the network structure (sometimes termed *rewiring*) in order to determine the desirability of certain structures. Agent capabilities include joining an existing team, initiating a new team, waiting, or rewiring the connections. If team formation is successful, participating agents become active in the task, complete the task, and then rejoin the set of available agents.

In previous work, sub-optimal coalitions have been studied in coalition formation with incomplete information and with limited computational resources [10, 11]. In an effort to reduce computational expense, agents can prune the list of possible coalition

partners to those considered most trustworthy or beneficial. These groups of agents who come together based on trust developed over past interactions are called congregations or clans [6]. By considering fewer potential coalition partners, as in our model, agents using clans can more easily compute possible coalitions within the time constraints imposed by the environment with the value of coalitions formed being good enough, or *satisficing* [10, 6].

This research implements a multi-agent system that uses an agent-organized network to facilitate coalition formation. The agents communicate tasks to neighboring agents to initiate the formation of a coalition to perform the task. When deciding which agents to include in the coalition, the agent forming the coalition considers each neighboring agent's skills. Agents share information to determine whom to connect to in the network. We examine the effect of greater amounts of information sharing on the overall performance of forming coalitions.

3.2. Agent-Organized Networks

In this research, we are concerned with decisions that improve coalition formation. How much information does an agent need in order to make decisions about rewiring, accepting a task, or proposing a given task? However, before we can answer these questions, we need to formally define our system.

An *Agent-organized network (AON)* is a set of inter-connected agents who collectively manage the structure of this network of agents by making individual decisions about which agent to connect to based on local information [4]. In our environment, *nodes* represent agents, and links represent social structure. The *neighbors* of an agent are all agents connected to it via a network arc. The *arcs* may represent a

variety of physical constraints, such as physical distance, limited communication, trust, or organizational hierarchy. *Teams* must be connected components in the social network.

Tasks are introduced to all agents within a region of interest, and are executed by agents who are connected in the network. A *task* is composed of a set of subtasks requiring specific skills. In our system, the total number of possible skills is denoted *SkillMax*. A task is specified by an array skillCt[1.. *SkillMax*] in which skillCt[i] indicates the number of agents possessing skill i that are required by the task. Thus, in an environment with five possible skills, a task with skillCt = [1,2,0,0,3] requires six agents: one with skill 1, two with skill 2, and three with skill 5.

In our system, skills are generated with a uniform distribution. We define the number of skills per task as *TaskSkills*, and the number of possible skills per agent as *AgentSkills* (with *AgentSkills*=1 for our tests).

In order to compare our results directly with those of [4], we performed identical tests. System parameters are summarized in Table 3.1. The length of the interval between task generations is *GenInterval*. If GenInterval = 1, one task is generated every iteration. If GenInterval = 5, one task is generated every five iterations. For simplicity, the number of skills required per task is constant, and all tasks have the same utility. Tasks are announced for a time interval *AnnouncePeriod*, which depends on the number of skills required. If the team for a task has not formed within AnnouncePeriod iterations, the task is removed from the system and counted as a failed task. If a team fails to form during a time interval of length *CommitTime*, an agent abandons the team. If all members abandon the team, the task returns to the available list (if its AnnouncePeriod has not expired). The duration of each task is specified as its *TaskDuration*.

Table 3.1. System parameters

System Parameter	Meaning	Default Value
SkillMax	Number of different skills in system	10
TaskSkills	Number of skills per task	10
AgentSkills	Number of skills per agent	1
GenInterval	Iterations between task generation	10
AnnouncePeriod	Number of iterations task is schedulable	10
CommitTime	Maximum time an agent will stay in a partially formed team	10
TaskDuration	Iterations required to complete task	10
DegreeLimit	Maximum degree of any node in system	20
RewireFrequency	Probability agent rewires in a given turn	1/N (N=# agents)

In forming teams, an agent may only communicate with its neighbors. Once a neighbor joins the team for a task, its neighbors are allowed to join the team. We term the neighbors of a neighbor, *FOAF* (friend of a friend) [2].

Each agent has a single skill, a position on the grid, and a neighbor set. Agents are in one of three states: committed, active, or uncommitted, as shown in Figure 2.1. A *committed* agent has joined a partial team. An *active* agent has joined a team that is now fully formed and is currently involved in completing the associated task for TaskDuration. An *uncommitted* agent is not associated with a team.

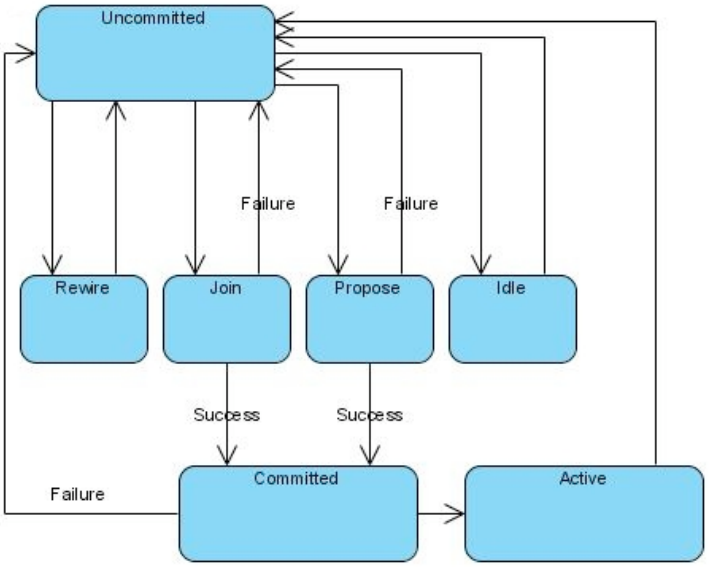

Figure 3.1. AON state diagram

Tasks require a team consisting of agents with a specific set of skills. A task consists of a taskID, an associated team (if any), an AnnouncePeriod, and a TaskDuration. A team consists of a taskID and a set of agents. A *complete team* has a match between required skills for the task and agents of the team. A *partial team* requires additional agents as some skill requirements are not met. Tasks are announced to agents within a fixed radius of the event. To eliminate variability in the results, in these tests, all tasks are known by all agents.

We randomly create networks in order to test our rewiring policies. We randomly assign locations and determine arcs based on physical distance. Arcs are initially assigned between nodes that are closer than a fixed parameter, but are changed during the process of rewiring. Since distance is not considered in the algorithm efficiency, this introduces no bias into the data. The degree of a node is limited to control communication. An

existing partial team can only be joined by an agent if one of its neighbors has already joined the team.

We often want to get a measurement of how many agents exist within a few links of an agent as more agents represent more potential coalition members. For this measurement, we define a local neighborhood. Agents within d network links of an agent are considered part of the agent's *local neighborhood*. Agents maintain a database of the skills possessed by the agents in their local neighborhood, as well as a database of skills lacking in the local neighborhood, determined by the number of failed tasks in which the skill was not filled by an agent.

At each iteration, an uncommitted agent can remain idle, join an existing partial team, rewire, or initiate a team for a task which is a member of its *EligibleTasks* set.

Selecting the best team to join: An agent selects an eligible task (if any) from tasks joined by its set of committed neighbors.

Selecting the best task to initiate: Rather than joining an existing team, an agent may choose to start a new team. A task is *viable* for an agent if its AnnouncePeriod has not expired, it requires a skill the agent possesses, and the task is not completed. A task is termed *unclaimed* if no partial or full team exists that is associated with the viable task. Only unclaimed tasks are considered for initiation.

Remaining idle: An agent may decide to remain idle as a strategy to remain available for tasks which do not yet involve one of the agent's neighbors or future proposed tasks.

Rewiring: The agent type controls adaptation method, which determines the conditions for rewiring, the frequency of rewiring, and the target of rewiring. Agents

choose the target for rewiring by communicating with the agents within the distance *communication depth* of network links.

3.3. Agent Types

An agent has the ability to remove a link to a neighbor and create a link to a new target. This rewiring is intended to increase the performance of the system. We define *system performance* as the fraction of tasks completed. We define *node performance* as the fraction of successful teams joined over the number of teams attempted by a node. We define *system efficiency* as the fraction of time an agent spends actively executing tasks. Agents executing tasks are termed *working*.

Rewiring involves an agent selecting one of its adjacent edges to disconnect from its current target and reconnect to another target. The edge is jointly owned by source and target, so either agent can initiate the rewiring, and no agent can refuse to be connected to a particular node (unless its maximum degree has been exceeded). We study four types of rewiring approaches: *structural, performance, egalitarian*, and *inventory*. Structural rewiring is motivated by findings in network topology [12]. Performance rewiring, defined in [4], is based on the desire to be connected to agents that have higher performance. Egalitarian agents attempt to utilize agents who are isolated from other agents in the network. Egalitarian rewiring is based on the desire to connect with agents with few neighbor connections. One goal is to improve system performance by making all agents more viable. A second goal is to associate with other agents who may have fewer options, increasing the likelihood of cooperating with these agents. Inventory rewiring seeks to connect to an agent with a skill that too few agents in the local

neighborhood possess. The goal of inventory rewiring is to broaden the number of tasks for which a viable team can be formed.

In performance adaptation, the adaptation trigger is based on local performance. Local performance is only valid when the number of teams joined is larger than a set number (five in our case). Local performance is computed as the number of successful teams joined divided by the number of teams joined. When an agent's local performance is less than the average of its neighbors' performance, rewiring is triggered: the agent removes the link to its worst performing neighbor and replaces it with a link to the best performing neighbor of its best performing neighbor.

In structural adaptation, rewiring is independent of performance and occurs with probability 1/N where N is the number of agents. An agent randomly picks a link and replaces it with a link to a FOAF based on preferential attachment. *Preferential attachment* means that the probability of a particular target is its degree divided by the total degree of all the neighbor's neighbors.

In egalitarian adaptation, rewiring also occurs win a probability of 1/N, where N is the number of agents. An agent seeks to connect to an agent who has the smallest number of neighbor connections.

In inventory adaptation, rewiring also occurs with a probability of 1/N, where N is the number of agents. An agent seeks to connect with an agent who possesses a skill that is the most needed in the local neighborhood. This skill is identified by utilizing the database the agent maintains based on the history of failed tasks encountered and skills possessed by agents in the local neighborhood.

3.4. Results

For our tests we examined the effect on the performance of each agent type when varying the communication depth from 2, to 10, and to 50. Recall, the *communication depth* is the number of links an agent can examine in each direction. To accomplish this, we do a breadth first search with the depth of the search equal to the communication depth.

In Figure 3.2, we see that with communication depth = 2, structural, egalitarian, and inventory agents have an increase in performance after adaptation begins at time step 1000, with structural and inventory agents showing the greatest improvement in performance. Egalitarian shows modest improvement after adaptation begins, and performance agents show very little improvement after adaptation begins. Inventory agents are limited in their ability to find agents possessing needed skills because of the small size of the local neighborhood with a communication depth of 2, but still manage to

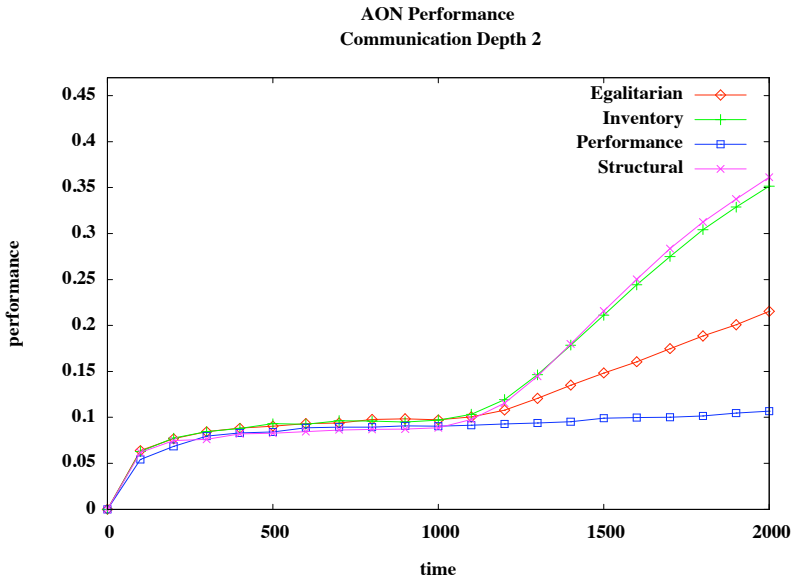

Figure 3.2. AON performance with communication depth = 2

nearly match structural agents' performance. Structural and egalitarian agents both show improved performance due to the structural changes that result in higher network density.

In Figure 3.3, we see the average density of the network for each agent type, where density is defined by the number of agents within n network links of an agent (we use n = 3 for computing density in all tests). High density indicates more agents in the local neighborhood. A high number of agents within a local neighborhood results in a better supply of skills to fill tasks encountered by the agents in the local neighborhood. Therefore, the performance will increase as density increases.

The higher average density resulting from egalitarian adaptation allows a higher level of performance than performance adaptation. Structural adaptation also results in a network that has a higher average density than networks with performance or inventory

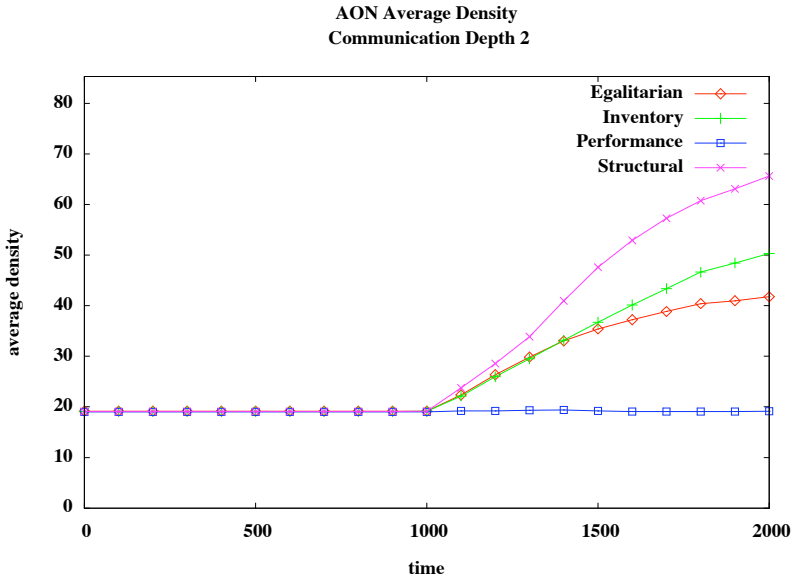

Figure 3.3. Average density with communication depth = 2

adaptation. Structural adaptation also has nodes with a high degree, as shown by the high maximum degree of structural agents shown in Figure 3.4, which we call hub nodes [1].

We use the term *hub node* to denote that the node has a large degree. We use the term *spoke node* to denote that the agent has a small degree but is connected to a hub node. Nodes that are neither hubs nor spokes are termed *limited*. These hub nodes result in a more centralized network, with a small number of hub nodes connected to a large number of spoke nodes.

This centralized structure allows structural adaptation to have both a higher network density and higher performance than egalitarian adaptation when the communication depth = 2.

In Figure 3.5 we see the results when the communication depth is increased to 10. All agent types show improvement compared to communication depth = 2. Structural

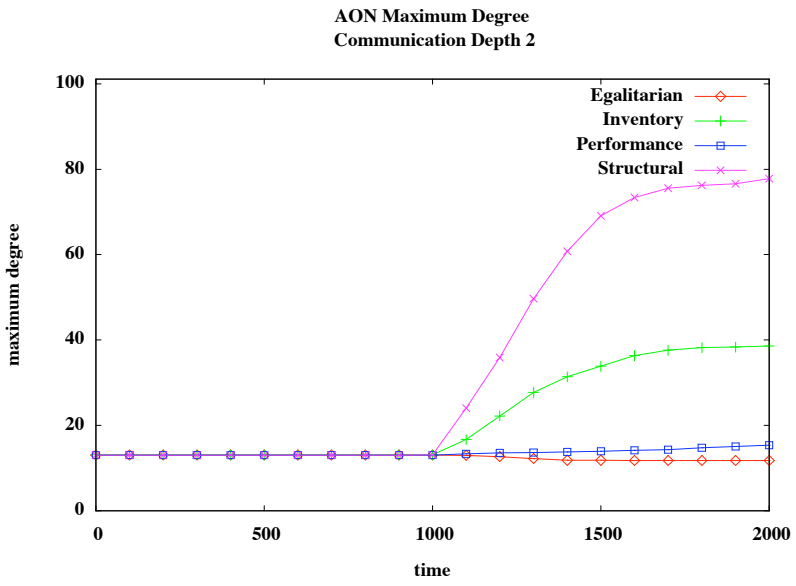

Figure 3.4. Maximum agent degree with communication depth = 2

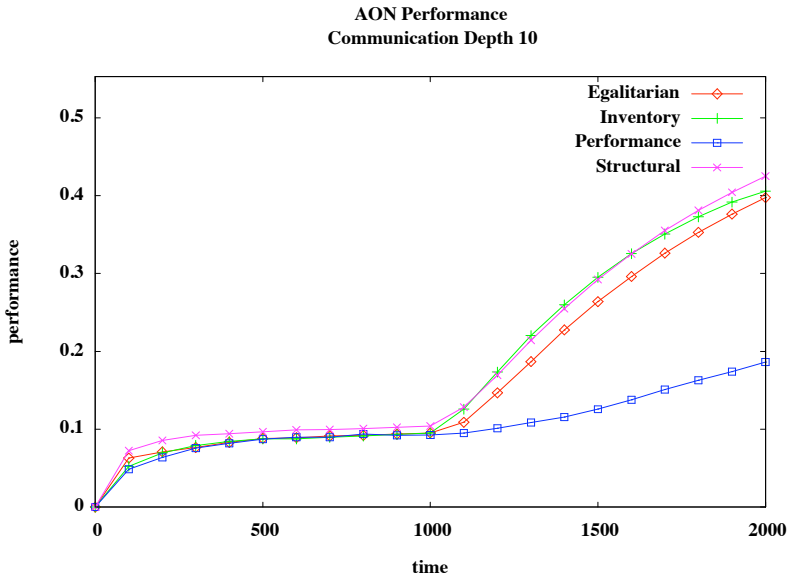

Figure 3.5. AON performance with communication depth = 10

agents improve from 0.36 to 0.43, performance agents improve from 0.11 to 0.19, egalitarian agents improve from 0.22 to 0.40, and inventory agents improve from 0.35 to 0.41 (see Figures 3.6 and 3.7).

Egalitarian agents have the highest average density, with structural agents the second highest. Egalitarian agents show a significant increase in average density and performance when compared with a communication depth of 2. Structural, inventory and performance agents show much smaller increases in performance and average density.

Once again, structural agents benefit from both the average density as well as a higher maximum degree indicating hub nodes. However, the average density of the egalitarian agents allows them to nearly catch up with the performance of the structural and inventory agents.

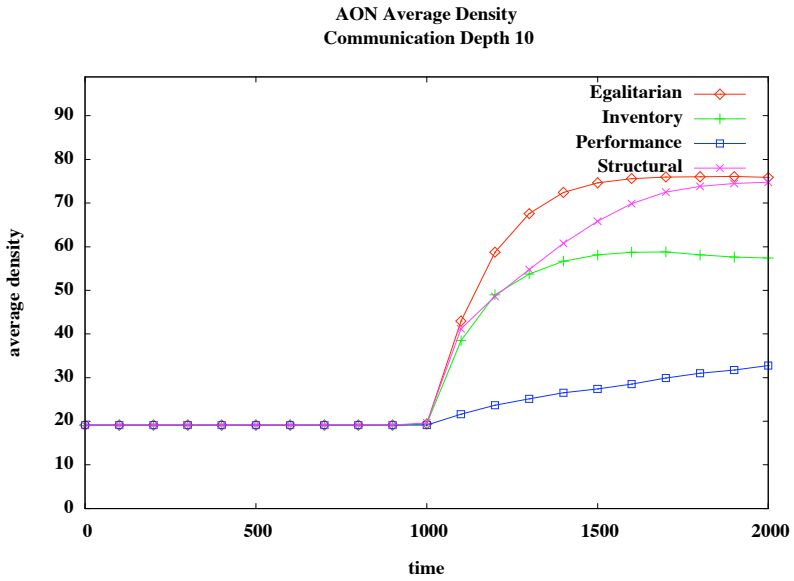

Figure 3.6. Average density with communication depth = 10

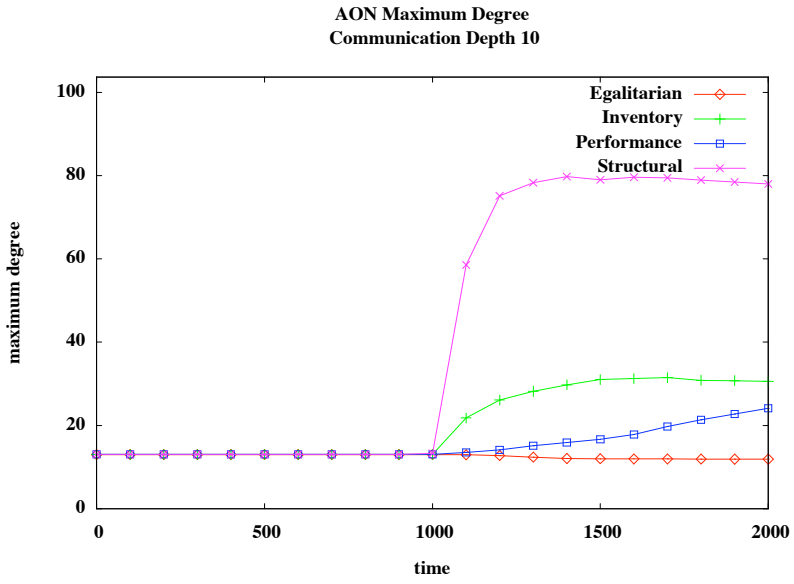

Figure 3.7. Maximum agent degree with communication depth = 10

In Figure 3.8, we see the results of increasing the communication depth to 50. Performance agents show the only significant improvement compared to communication depth = 10. Structural agents remain at 0.43, performance agents improve from 0.19 to 0.27, egalitarian agents improve from 0.40 to 0.41, and inventory agents improve from 0.41 to 0.42.

Structural, egalitarian, and inventory agents show significantly less improvement in performance when increasing to communication depth = 50. This comes because performance agents are the only agent still showing an increase in network density compared to communication depth = 10.

As shown in Figure 3.9, structural agents for the first time achieve the highest average density. With the maximum degree remaining the same with all three tests, structural agents increasingly rely on a higher average density to achieve higher performance.

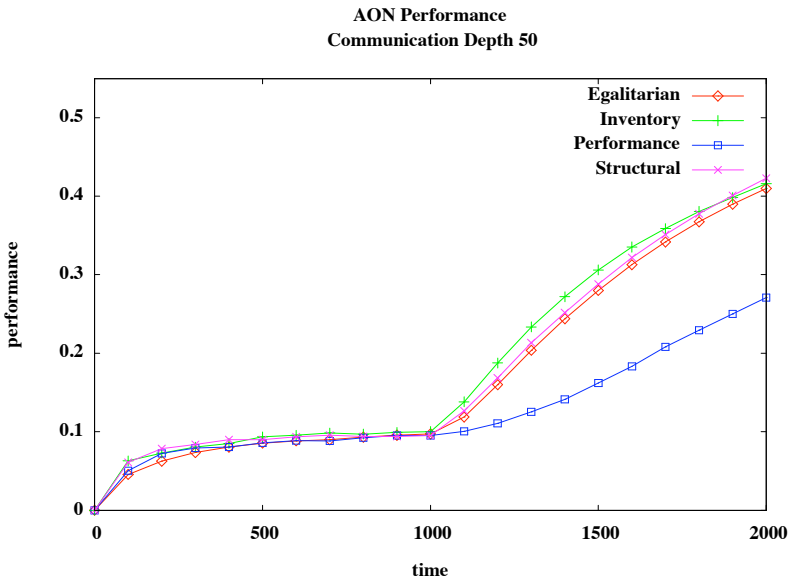

Figure 3.8. AON performance with communication depth = 50

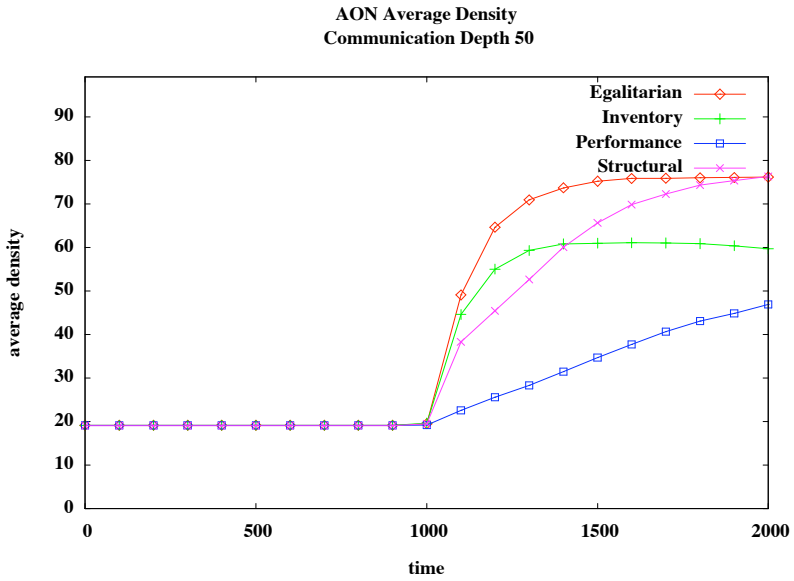

Figure 3.9. Average density with communication depth = 50

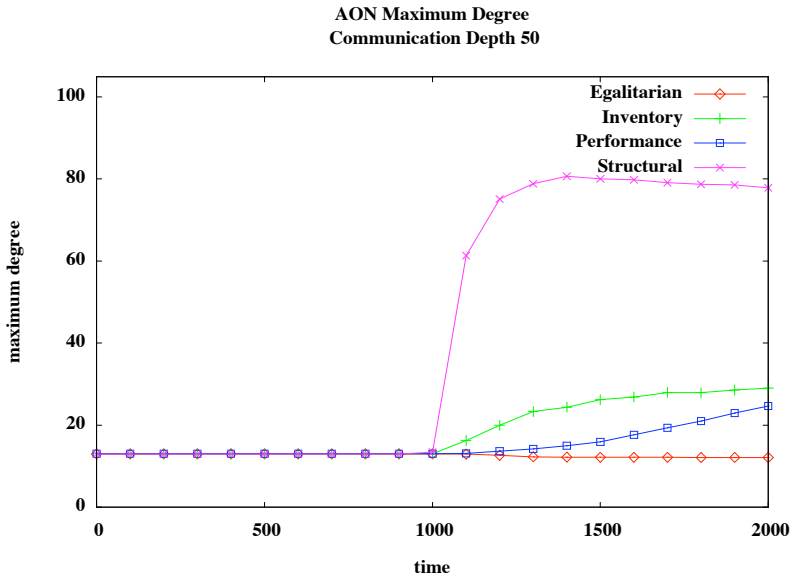

Figure 3.10. Maximum agent degree with communication depth = 50

As the communication depth increases, as shown in Figure 3.11, we see the effects of the increasing amount of information available when adapting the network's connections. Structural and inventory agents are the most sensitive to communication depth, with their performance increasing much faster as communication depth increases, followed by egalitarian agents, then performance agents. As communication approaches complete information sharing, performance agents have greater increases in performance, but still do not reach the levels of performance of the other agents. The other agents reach nearly the same level of performance when there is complete information sharing, but structural and inventory agents are able to reach higher levels of performance the fastest. Therefore, all agents benefit from complete information sharing, but inventory and structural agents perform the best regardless of the level of information sharing.

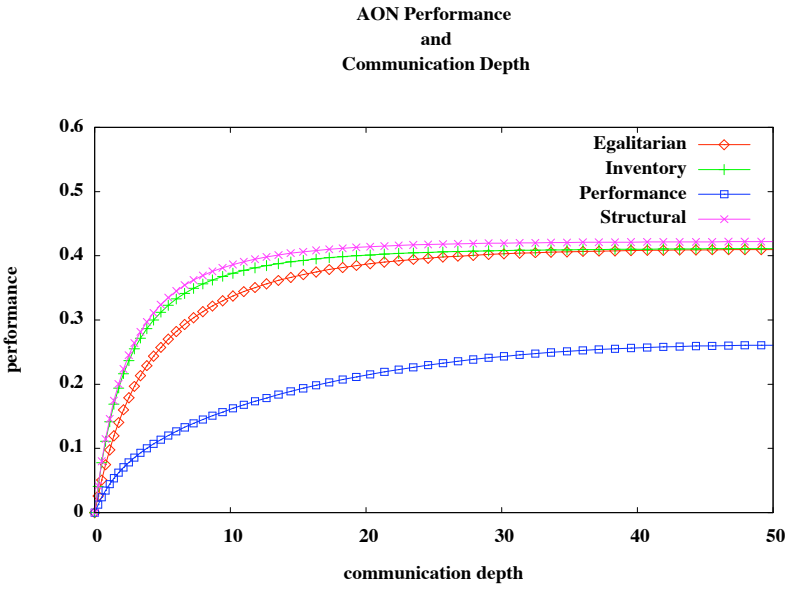

Figure 3.11. AON performance vs. communication depth

3.5. Previous Work

Gaston and des Jardins [4] implement multiagent team formation in which tasks are globally known. Gaston and des Jardins rewire based on performance and structural metrics. Under performance adaptation, an agent whose performance is less than the performance of its neighbors is allowed to reconnect to the best of the neighbor's neighbors. Under structural adaptation, agents rewire to select a target from the set of neighbor's neighbors. The probability of connecting to an agent is proportional to a node's degree. Our results show the performance of Gaston and des Jardins' performance and structural agents can be improved upon by making strategic task selections and by using task impatience.

Barton and Allan [1] implement strategic task selection and impatient task waiting in an agent-organized network augmenting the performance of agents presented in [4]. Limits are placed on maximum agent degree, and the effects of choosing a degree limit are examined. High task loads are shown to decrease performance in all agents, but strategic and impatient agents are shown have higher performance levels than other strategies. Reasonable degree limits are shown not to have a significant impact on performance.

Soh and Tsatsoulis [10, 11] present their method of creating sub-optimal coalitions in a dynamic environment to deal with partial information and meet time constraints imposed by the real-time nature of the environment. They apply their method in a distributed target-tracking environment wherein the agents are stationary. Our method is based on their work, with an added challenge of forming coalitions among agents who navigate within a dynamic environment.

Griffiths and Luck [5] discuss their method of coalition formation for clans whose cooperation is of medium-term duration. They combine agent motivations and agent trust to determine when to create and terminate these clans. They introduce a kinship motivation, wherein agents determine the probability of encountering tasks that require cooperation based on tasks recently encountered.

Soh and Chen [9] define a collaboration utility between agent-based history of successful collaboration and measures the reliance of one agent on another in forming coalitions. While no discrete labeling of clan is used, the collaboration utility specifies a loose clan in which trusted agents are more likely to be recipients of proposals and are more likely to be accepted.

3.6. Conclusion

We have examined the effects of increasing the amount of information available to agents when adapting their network connections by comparing agents using different adaptation methods while building teams in an agent-organized network. As the communication depth increases, the amount of information an agent has about the local neighborhood increases, allowing the agent to choose the best agent to attach to in the network.

When agents develop higher density networks, they are better able reach higher levels of performance. This results from an increase in the number of skills available in their local neighborhood. This increase in available skills leads to a higher probability of having a sufficient number of agents possessing the skills of a task encountered by the agents in the local neighborhood, leading to higher performance.

All agents have been shown to prefer complete information sharing among agents, with structural agents outperforming all other agent types when both the least information is shared as well as when complete information is shared. Inventory agents have been shown to outperform egalitarian agents when smaller amounts of information are shared, with egalitarian agents matching inventory agents' performance when complete information is shared. Inventory agents compare well with structural agents, nearly matching structural agents' performance despite a lower network density and lower maximum degree.

References

[1] L. Barton and V. H. Allan Methods for coalition formation in adaptation-based social networks. In *CIA)*, 2007, 285–297.

[2] D. Brickley and L. Miller. (Nov, 2007) FOAF: The 'friend of a friend' vocabulary. Available online at http://xmlns. com/foaf/0.1/.

[3] V. Dang and N. Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *AAMAS*, 2004, 564–571.

[4] M. E. Gaston and M. des Jardins. Agent-organized networks for dynamic team formation. In *AAMAS*, 2005, 230–237.

[5] N. Griffiths and M. Luck. Coalition formation through motivation and trust. In *AAMAS*, 2003, 17–24.

[6] B. Horling and V. Lesser. A survey of multi-agent organizational paradigms. *Knowledge Engineering Review,* 19:281–316, 2005.

[7] H. C. Lau and L. Zhang. Task allocation via multi-agent coalition formation: Taxonomy, algorithms and complexity. In *15th ICTAI*, 2003, 346–350.

[8] T. Sandholm and V. Lesser. Coalitions among computationally bounded agents. *Artificial Intell., Special Issue on Economic Principles of Multi-Agent Systems,* 94:99–137, Jan. 1997.

[9] L.-K. Soh and C. Chen. Balancing ontological and operational factors in refining multiagent neighborhoods. In *AAMAS*, 2005, 745–752.

[10] L.-K. Soh and C. Tsatsoulis. Utility-based multiagent coalition formation with incomplete information and time constraints. In *IEEE International Conference SMC,* vol. 2, 2003, 1481–1486.

[11] L.-K. Soh and C. Tsatsoulis. Reflective negotiating agents for real-time multisensor target tracking. In *IJCAI*, 2001, 1121–1127.

[12] H. P. Thadakamalla, U. N. Raghaven, S. Kumara, and R. Albert. Survivability of multiagent-based supply networks: A topological perspective. *IEEE Intell. Syst.,* 19:24-31, 2004.

CHAPTER 4

ADAPTING TO CHANGING RESOURCE REQUIREMENTS FOR COALITION FORMATION IN SELF-ORGANIZED SOCIAL NETWORKS[3]

Abstract

Coalition formation in social networks consisting of a graph of interdependent agents allows many choices of tasks to select and with whom to partner in the social network. Agents communicate with agents within n network links in their surrounding network. These agents are considered part of an agent's local neighborhood. Agents maintain a database of skills possessed by agents in their local neighborhood. We compare agents of three different types. Structural agents seek to create a scale-free network. Egalitarian agents seek equal distribution of connections among agents, resulting in a dense network structure. Inventory agents seek to connect to agents who possess a skill not found in their current local neighborhood. We examine the ability of the agents to deal with static skill demand patterns, changing skill demand patterns, and a mismatch of the skills supplied to the skills demanded.

4.1. Introduction

This research implements a multi-agent system that uses an agent-organized network to facilitate coalition formation. The agents communicate tasks to neighboring agents in the network to initiate the formation of a coalition to perform the task. When deciding which agents to include in the coalition, the agent forming the coalition considers information about the local neighborhood. Agents use several strategies to

[3] Coauthored by Levi Barton and Vicki Allan.

decide how to rewire the social connections. Previous work has used a uniform skill distribution. This is not as realistic or as interesting as a non-uniform skill distribution.

Task allocation in multiagent systems has been studied by many researchers with different assumptions and emphases. Our work builds off previous research. In [1], agents are organized in a network, and agents only interact with immediate neighbors – termed the *social network*. Like our method, agents self-organize, and the delay in completing the task is an important consideration. They learn to determine both when to reorganize and how to pick a new neighbor. Both the network and the policy for forming coalitions are changing. However, Abdallah and Lesser restrict their problem to that of task allocation (assigning one agent to do a task) rather than coalition formation. Gaston, desJardins, and Bulka [4, 6] consider social networks and task formation with multiple skills per task, but do not have varying agent types.

However, most of the research does not consider social networks, and studies the problem using a centralized algorithm. Kraus [7] assumes a centralized protocol in which each agent knows the capabilities of all others and there is no network limiting which agents can interact in seeking to complete tasks. Manisterski et al. [9] consider task allocation in cooperative and noncooperative settings. Others consider distributed algorithms, but no social network [8, 12]. Sander [10] considers a domain in which agents can move to get to a task and all agents have the same skill set. Shehory [11] proposes a way to locate agents to complete the tasks. Each agent keeps track of other agents they know, and negotiation is based on lattice-like graphs. Our work uses a social network whose topology is unrestricted. Easwaran and Pitt [5] study complex tasks in a supply chain. Walsh and Wellman [14] look at dependence chains between tasks.

Sugawara et al. [13] use a social network to find a server, but have coalitions of size one. They use various strategies for picking a server to perform the task. One such strategy penalizes an agent that cannot accept the task so that it is not automatically selected again.

In [15], Weerdt et al. consider allocating tasks in a social network. Agents have multiple resources they can contribute to the task. Task allocation must obey social relationships. Tasks are given to a particular agent who is responsible for deciding if the task should be done and, if it should be done, finding other agents within the social network that can complete the necessary resource requirements. They make decisions based on *maximum efficiency*, which is defined as the utility of the task divided by the total resources required. They compare efficiency between their greedy algorithm and the optimal solution. Optimal is computed using a network flow graph over all subsets of tasks. They look at each resource separately (to see if all the tasks in the subset can be satisfied) and then succeed if every resource succeeds separately. Obviously, this cannot be done for a large number of tasks, but it serves as a valuable comparison for small systems. They find that "having more connections means getting better results" [15:505].The quality of the solution using greedy distributed allocation protocol depends on both the resource ratio and the network degree. This echoes our findings.

Our method builds on the ideas of coalition formation and task allocation as applied to uniform and non-uniform skill distribution in the tasks and the agents. We examine the effects of changing the skill distribution of skills possessed by agents and the distribution of skills in the tasks that must be completed.

4.2. Agent-Organized Networks

An *Agent-organized network (AON)* is a set of inter-connected agents who collectively manage the structure of a network of agents by making individual decisions about which agent to connect to based on local information [6]. In our environment, nodes represent agents, and links represent social structure. The *neighbors* of an agent are all agents connected to it via a network arc. Agents are allowed to join an existing task, propose a new task, delay taking any action, or change a link to one of its neighbors in the network. We term this changing of a link in the social network *rewiring*.

Tasks are introduced to all agents within a network neighborhood, and are executed by agents who are connected in the network. A task is composed of a set of subtasks requiring specific skills. Agents within three network links of an agent are considered part of the agent's *local neighborhood*. Agents maintain a database of the skills possessed by the agents in their local neighborhood, as well as a database of tasks lacking in the local neighborhood, determined by the number of failed tasks in which the skill was not filled by an agent.

We study agents, which we introduced in [3], wherein agents use one of three types of rewiring approaches: *structural, egalitarian*, and *inventory*. Structural agents seek to connect to agents with the most network connections [6]. Structural rewiring results in a centralized network structure consisting of a small number of high-degree *hub* nodes surrounded by a larger number of low-degree *spoke* nodes.

Egalitarian agents attempt to connect to agents with few network connections. Egalitarian rewiring is based on the desire to equally distribute the number of network connections among agents in the network. The goal of egalitarian rewiring is to improve

system performance by giving all agents an equal opportunity to participate in tasks by minimizing the number of isolated agents.

Inventory agents seek to connect to an agent who possesses a skill that is the most needed in the local neighborhood. This skill is identified by querying the local database for the skill that is in low supply among the agents in the local neighborhood. Among those skills, the database is again queried to determine which of these skills has caused the most failed teams.

4.3. Skill-Supply and Skill-Demand

In these tests, we studied the ability of the various rewiring methods to deal with tasks that exhibit specific patterns of skills usage. Previous tests [2, 3, 6] have assumed that skills are assigned to agents in a uniform manner and that all skills are equally likely in a task. Uniform skill-supply allows agent strategies which simply try to get a larger number of neighbors (structural) or rescue agents with few neighbors by befriending them (egalitarian) to be fairly successful. However, assuming that skills within a task are equally likely is not nearly as interesting (or realistic) as assuming there are some combinations of resources that are commonly required together.

We use the term *skill-supply* to describe the skill congregation used to assign skills among the agents in the system. We use the term *skill-demand* to describe the skill congregation used when creating tasks in the system. When the skill-supply and the skill-demand are of the same congregation type, we term this a *skill-match*. We look at the effect of combinations of skill-supply and skill-demand on the performance of inventory, structural, and egalitarian agents.

We examine three types of skill congregation: congregate, overlapping, and uniform. Congregate skill structure consists of three disjointed sets of skills that occur together in any one task.

As can be seen in Figure 4.1, skills 0, 1, and 2 occur together, skills 3, 4, and 5 occur together, and skills 6, 7, 8, and 9 occur together. The skill weights for each skill in Figure 4.1 represent the probability of a skill to occur in a task produced by a congregation of skills. For example, the quantity of skill 2 required in tasks produced by Skill Congregation 1 in Figure 4.1 is three times more than the quantity of skill 1 required in tasks produced by Skill Congregation 1.

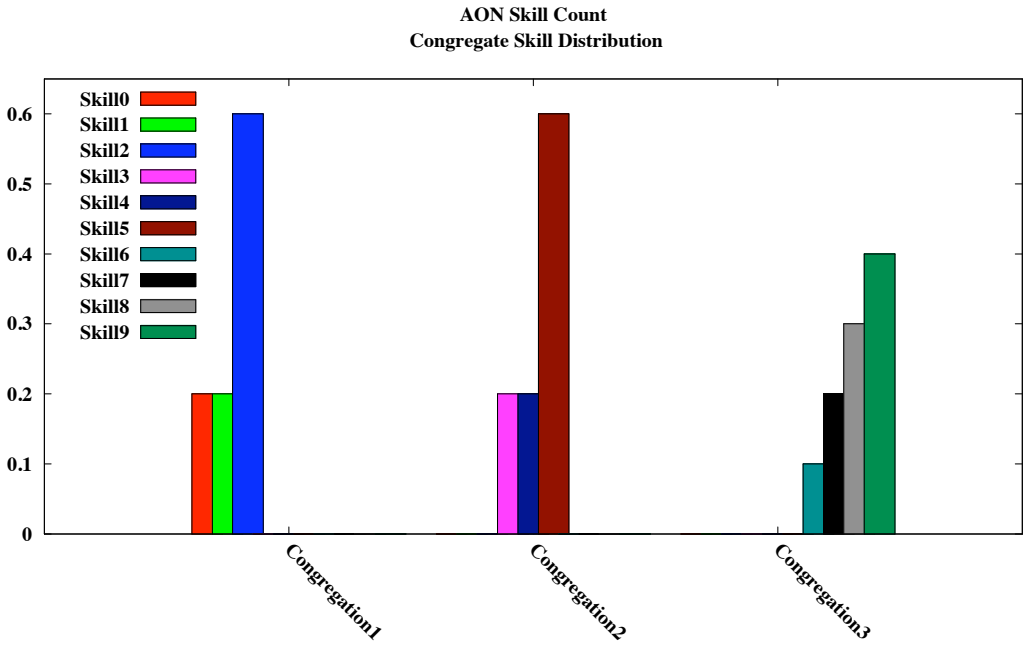

Figure 4.1. Congregate skill congregation

In Figure 4.2, we see overlapping skill congregation wherein the skills in each congregation no longer form a partition. There are more skills in each congregation compared to Figure 4.1. Thus, the skill weights are smaller, resulting in smaller differences in the probability that each skill is required by a task.

With uniform skill congregation (see Figure 4.3), all skills have the same skill weight; therefore, each skill has the same probability of being included in a task.

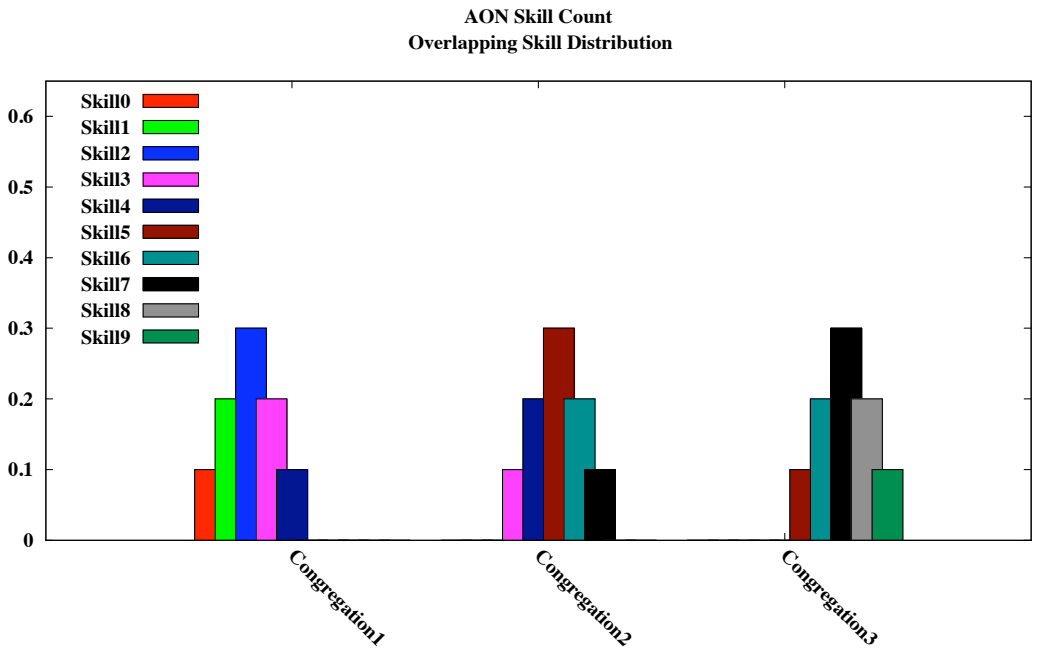

Figure 4.2. Overlapping skill congregation

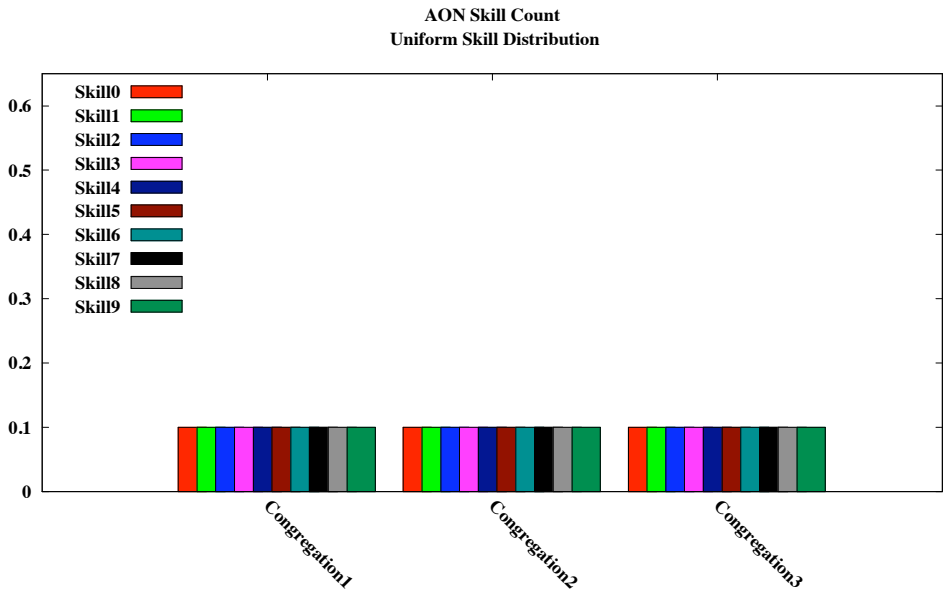

Figure 4.3. Uniform skill congregation

4.4. Results

The first group of tests deals with the ability of the agents to perform well when skill requirements for a task are determined by congregate, overlapping, and uniform skill congregations. We present the results of the three agent types as a function of the congregation factor. The congregation factor varies from uniform skill congregation (skills are randomly assigned to tasks) to congregate skill congregation (the sets of skills used together belong to disjoint partitions of the skill set).

Each set of results were run twenty times, resulting in a confidence interval of 0.01, giving statistical significance to our results. The tests were all run for 4000 time steps with rewiring beginning at time step 0. The tests were run with 100 agents, each possessing one skill. Each task introduced had 10 skills to be completed.

4.4.1. Constant Skill-Demand

The first set of tests was run with static skill-supply and skill-demand for the entire test. These tests show which skill-demand and skill-supply conditions lead to the best performance for each agent type.

4.4.1.1. Inventory Agents. Table 4.1 shows the performance levels inventory agents reach after 4000 time steps for each combination of skill-supply and skill-demand.

When there is congregate skill-supply, inventory agents prefer congregate skill-demand, but they still outperform structural and egalitarian agents when there is overlapping or uniform skill-demand (as will be seen in future sections).

When there is overlapping skill-supply, inventory agents prefer overlapping skill-demand, but they still outperform both structural and egalitarian agents with either congregate or uniform skill-demand (as is seen by comparing performance shown in this section with that shown in other sections).

When there is uniform skill-supply, as we see in Figure 4.4, inventory agents prefer overlapping skill-demand, followed by congregate skill-demand. Inventory agents outperform structural and egalitarian agents when skill-demand is congregate. Inventory agents outperform structural and egalitarian agents when skill-demand is overlapping and when

Table 4.1. Inventory agent performance at time step 4000

	congregate supply	overlapping supply	uniform supply
congregate demand	0.59	0.48	0.64
overlapping demand	0.34	0.59	0.66
uniform demand	0.39	0.40	0.53

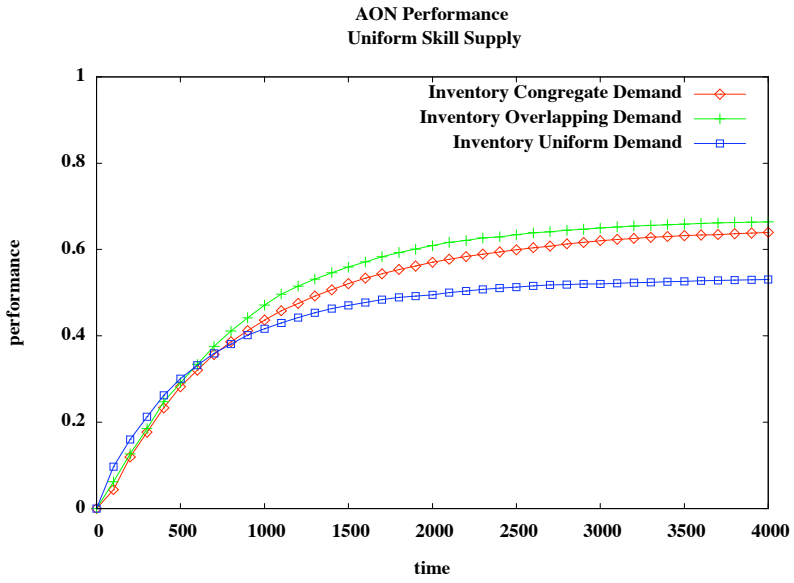

Figure 4.4. Inventory agent performance with uniform skill-supply

there is uniform skill-supply. However, structural agents outperform inventory agents when there is uniform skill-demand and uniform skill-supply.

Inventory agents do better in more combinations of skill-supply and skill-demand than either structural or egalitarian agents. This illustrates the benefit of using inventory rewiring. In Table 4.1, the number of skill-supply and skill-demand combinations wherein inventory agents are able to achieve performance of 0.5 or better is five of the possible nine combinations. In Table 4.2, structural agents have three of the possible nine combinations of skill-supply and skill-demand wherein performance was 0.5 or better. In Table 4.3, egalitarian agents have no combinations of skill-supply and skill-demand that have performance of 0.5 or better.

4.4.1.2. Structural Agents

Table 4.2 shows the performance levels which structural agents reached after 4000 time steps for each combination of skill-supply and skill-demand.

Table 4.2. Structural agent performance at time step 4000

	congregate supply	overlapping supply	uniform supply
congregate demand	0.38	0.29	0.37
overlapping demand	0.22	0.52	0.50
uniform demand	0.30	0.37	0.58

When there is congregate skill-supply, structural agents prefer congregate demand. However, even with a skill-supply and skill-demand match, the performance level is less than 0.5.

When there is overlapping skill-supply, structural agents prefer overlapping skill-demand. The performance level matching overlapping skill-supply and overlapping skill-demand is greater than 0.5, but still less than the performance of inventory agents for the same skill-supply and skill-demand.

When there is uniform skill-supply, structural agents prefer uniform skill-demand (see Figure 4.5). Structural agents outperform both egalitarian and inventory agents when both skill-supply and skill-demand are uniform, showing a strong preference for uniform skill congregation.

These results show the structural agent has a preference for either a match between the type of skill-supply and the type of skill-demand, or uniformity between skill-supply or skill-demand. This is because structural agents ignore skills that are in high demand or in low supply, seeking only to have a large number of immediate neighbors in the network. Therefore, structural agents perform best when there is a close

match in skill-supply and skill-demand, giving any large group of agents the best chance to fill the skills of tasks needed to be completed.

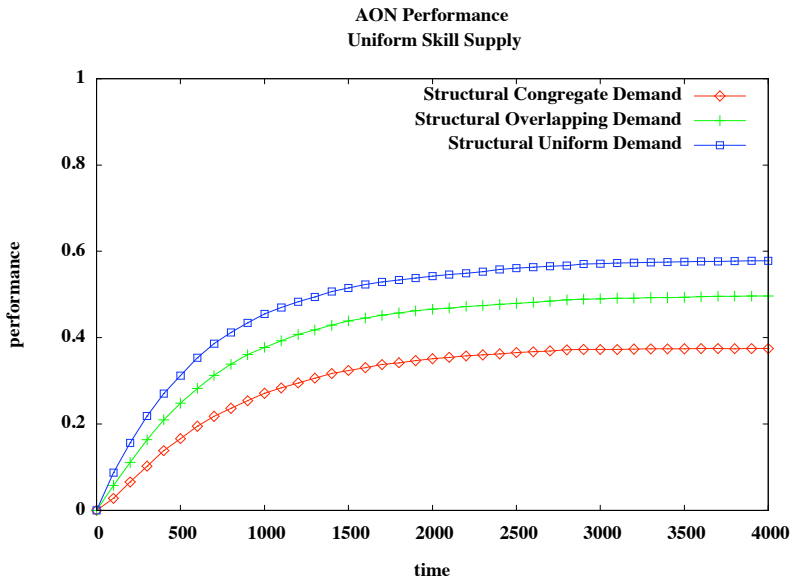

Figure 4.5 Structural agent performance with uniform skill-supply

4.4.1.3. Egalitarian Agents

Table 4.3 shows the performance levels egalitarian agents reached after 4000 time steps for each combination of skill-supply and skill-demand.

When there is a congregate skill-supply, egalitarian agents prefer congregate skill-demand. However, the performance is less than either inventory or structural agents.

When there is overlapping skill-supply, egalitarian agents prefer overlapping skill-demand. Once again, the performance is less than either inventory or structural agents.

When there is uniform skill-supply, egalitarian agents prefer uniform skill-demand (see Figure 4.6). The performance is 0.43; the closest egalitarian agents get to

reaching performance of 0.5. However, the performance is, once again, less than that of inventory or structural agents.

Table 4.3. Egalitarian agent performance at time step 4000

	congregate supply	overlapping supply	uniform supply
congregate demand	0.25	0.22	0.19
overlapping demand	0.08	0.30	0.28
uniform demand	0.13	0.17	0.43

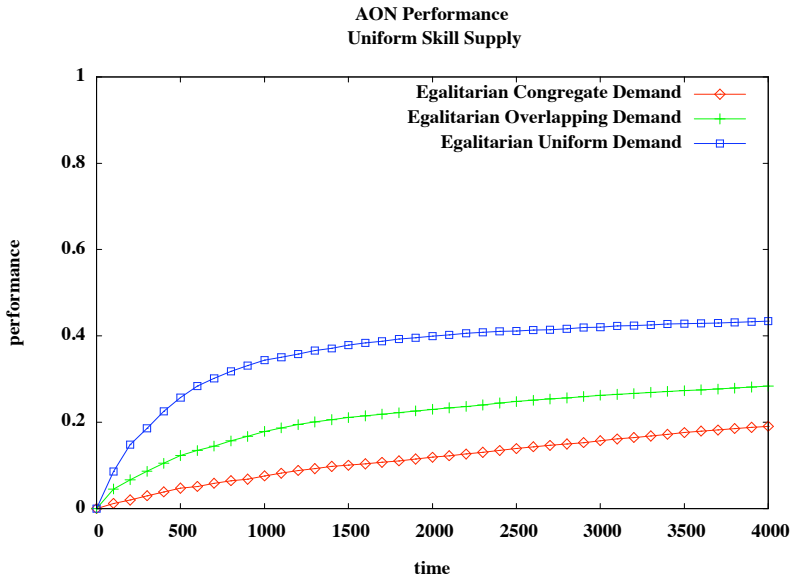

Figure 4.6. Egalitarian agent performance with uniform skill-supply

Egalitarian agents, like structural agents, have a preference for having a match between skill-supply and skill-demand. However, egalitarian agents lack the network density to compete with the performance levels reached by structural agents. The goal of egalitarian rewiring is to give all agents an equal chance at belonging to a coalition.

When the requirements for tasks are easy to satisfy, this strategy is beneficial. Let T_p be the number of simultaneous tasks that are executed in practice, and let T_i be the number of simultaneous tasks that are executed if all agents can be used. When T_p approaches T_i, egalitarian agents do well. Note that for structural agents, T_p is limited by the number of hub nodes. Structural agents are smart enough to create enough hub nodes to achieve T_p. Egalitarian agents have no such flexibility. However, when T_p is significantly less than T_i, egalitarian agents are hurt because they have lessened the ability to form T_p coalitions. They are literally "spread too thin."

4.4.2. Changing Skill-Demand

The second group of tests deals with the ability of the agents to adapt to a change in the skill-demand in the middle of a test. This change requires the agents to switch from satisfying tasks that have sets of skills that occur together to tasks that have new sets of skills that occur together. We want to understand how each agent type dynamically adapts to such changes. These tests were run for 4000 time steps with rewiring starting at time step 0, and the skill-demand was changed at time step 1000. We will now examine the effects this change in skill-demand has on agent performance.

4.4.2.1. Congregate Skill-Supply. We define congregate skill-supply as being three sets of skills that occur together in tasks. These sets of skills have a different weight for the probability of times each skill in the set occurs in a task produced by that set, or cluster of skills, as shown in Figure 4.1.

When there is congregate skill-supply, inventory agents always reach time step 4000 with a higher level of performance than structural or egalitarian agents. This is because inventory agents perform best when the distribution of skills in the skill-demand

and the skill-supply is not evenly distributed for all skills. Inventory agents look for skills that are needed more than others, requiring a skill shortage. When there is congregate skill-supply, this shortage is seen as a result of some skills being in greater supply than others. Therefore, inventory agents can help compensate for this shortage in supply by seeking to connect to agents who possess the skills for which scarcity exists.

When there is congregate skill-supply, inventory agents struggle when switching from congregate skill-demand to either overlapping or uniform skill-demand. This can be seen in Figure 4.7, wherein inventory agents achieve their highest level of performance just before the switch from congregate to uniform skill-demand occurs. After the switch occurs, inventory agents decrease in performance, ending at time step 4000 with a small lead in performance over structural agents. The results for switching from congregate skill-demand to overlapping skill-demand are similar. This shows inventory agents are unable to react to a change from the preferred skill-demand to another, due to the sudden lack of clarity in deciding which skill is in greatest shortage. With congregate skill-demand, inventory agents can always find a skill that is in shortage that will help ensure

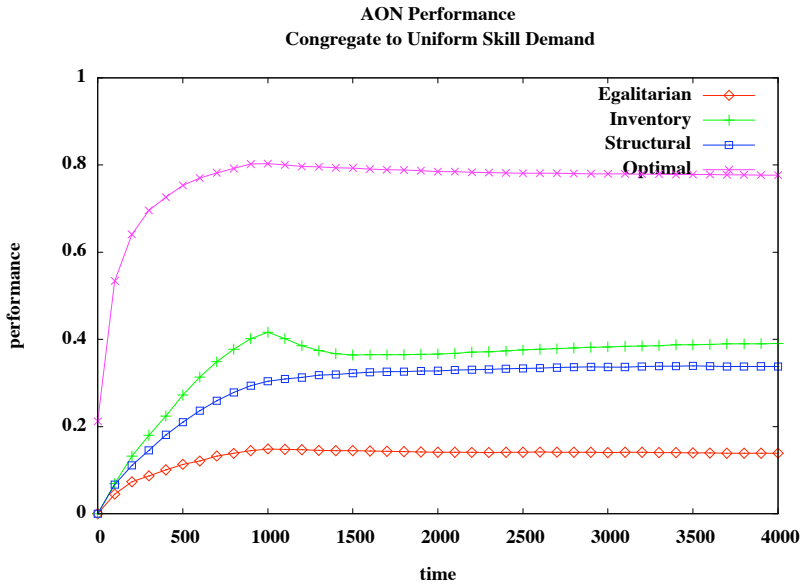

Figure 4.7. Congregate skill-supply, congregate to uniform skill-demands

the success of future coalitions. With overlapping and uniform skill-demands, inventory agents are faced with less difference in the demand for an individual skill, making it harder to improve future coalition formation by adding one targeted skill to the local neighborhood.

4.4.2.2. Overlapping Skill-Supply. We define overlapping skill-supply as being three congregations of skills that occur together in tasks. These sets of skills have a different weight for the probability of each skill in the set occurring in a task produced by that congregation of skills, as shown in Figure 4.2. There is less difference between individual weights for skills in each skill congregation when compared to congregate skill-supply.

When there is overlapping skill-supply, inventory and structural agents have the same level of performance at time step 4000 in all cases except for the following:

1) when skill-demand changes from either uniform or overlapping skill-demand to congregate skill-demand wherein inventory agents reach a higher level of performance; and

2) when skill-demand changes from uniform to overlapping skill-demand wherein structural agents reach the highest level of performance.

As shown in Figure 4.8, inventory agents reach their highest level of performance just before the change from congregate to uniform skill-demand. After the switch, the performance of inventory and structural agents is the same.

4.4.2.3. Uniform Skill-Supply. We define uniform skill-supply as three sets of skill congregation wherein all skills have an equal probability of being included in a task. This can be seen in Figure 4.3, wherein the skill weights are equal for all ten skills.

When there is uniform skill-supply, inventory agents have a higher performance level than structural agents at time step 4000 in all cases — except for the following cases:

1) when skill-demand changes from overlapping to uniform, where inventory and structural reach the same performance level; and

2) when skill-demand changes from congregate to uniform, where inventory agents reach a lower performance level than structural agents. See Figure 4.9.

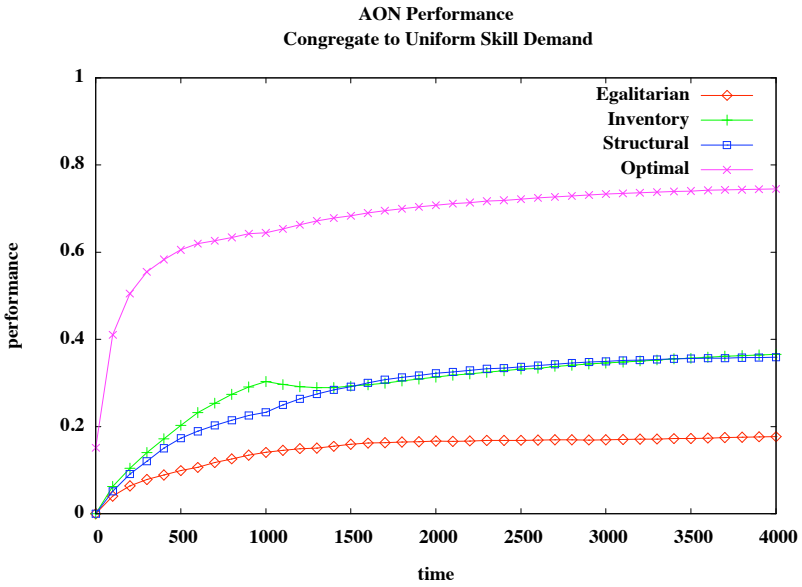

Figure 4.8. Overlapping skill-supply, congregate to uniform skill-demands

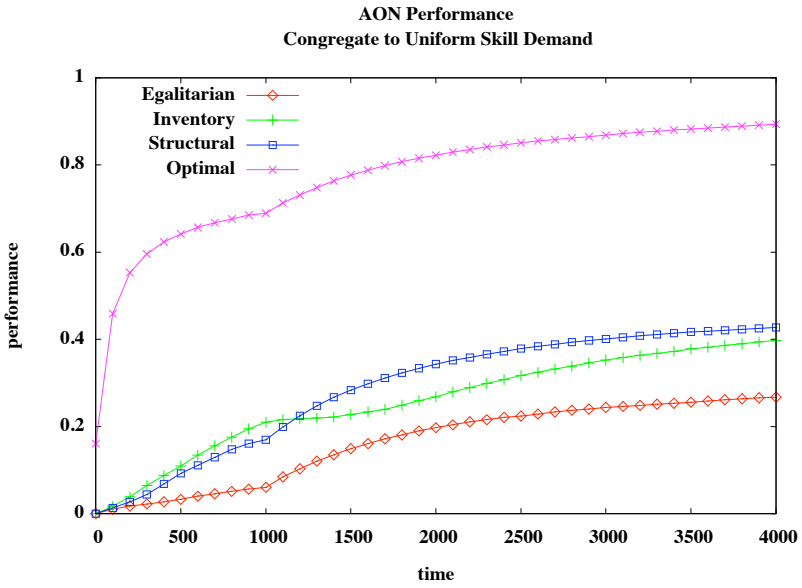

Figure 4.9. Uniform skill-supply, congregate to uniform skill-demands

4.4.3. Results Summary

Our tests show inventory agents perform well compared to structural and egalitarian agents in a variety of skill-supply and skill-demand environments. In a static skill-demand environment, inventory agents outperform egalitarian agents in all combinations of skill-supply and skill-demand, and they outperform structural agents in all but one skill-supply skill-demand combination (uniform skill-supply and uniform skill-demand).

In a dynamic skill-demand environment, inventory agents show a preference for congregate skill-demand and struggle when changing to uniform skill-demand. When congregate skill-supply is used, inventory agents outperform the other agents in all changes between skill-demand. When overlapping skill-supply is used, inventory agents tie structural agents in performance when changing to uniform skill-demand, and do worse when switching from uniform skill-demand to overlapping skill-demand. When uniform skill-supply is used, inventory agents tie or perform worse than structural agents when skill-demand changes to uniform skill-demand.

These results show inventory agents have a preference for congregate skill-demand and struggle with uniform skill-demand. This is caused by the lack of significant difference in the demand for any one skill in uniform skill-demand. Inventory agents rely on singling out a skill that has a higher demand than other skills. This is not possible with uniform skill-demand since all skills have the same level of demand.

4.5. Conclusion

An agent-organized network gives each agent the opportunity of choosing who its neighbors will be. Each type of agent uses different criteria in choosing which agent

would be most desirable to add to the set of existing neighbors. Structural agents choose the agent that is connected to the most neighbors. This results in a hub network structure wherein a few agents are at the center of hubs, with a majority of the agents having a smaller number of network connections along the outside of the hubs. Egalitarian agents choose the agent who has the fewest connections to neighbors. This results in a network structure wherein agents have an equal number of connections to neighbors. Inventory agents choose the agent who possesses the most needed skill. The most needed skill is determined by maintaining a count of each type of skill available nearby in the network, and by keeping a count for each skill wherein the particular skill was unfilled in a failed task. Inventory agents build a network neighborhood that is able to conform to the skill distribution predicted to be faced in future tasks based on the skill distribution encountered in the past. Thus, inventory agents are better suited to all skill-supply and skill-demand environments.

We have shown that inventory agents outperform structural and egalitarian agents in nearly all constant skill-demand environments. Structural agents outperform inventory agents in a constant skill-supply environment only when there is uniform skill-supply and uniform skill demand. Inventory agents do not perform as well with uniform skill-demand due to the lack of clarity in deciding which skill is most needed. Since all skills are uniformly distributed in uniform skill-demand, no one skill can be targeted as being in short supply.

We have also shown inventory agents outperform structural and egalitarian agents when there is congregate skill-supply in a changing skill-demand environment. Inventory agents do not outperform structural agents for all changing skill-demand environments

when there is overlapping or uniform skill supply. However, inventory agents still outperform structural and egalitarian agents when switching to congregate skill supply for both overlapping and uniform skill-supply environments.

We have shown the benefits of using inventory agents to form social networks of agents capable of dealing with non-uniform skill congregation. We have also shown inventory agents capable of dealing with a change in skill congregation. However, inventory agents struggle when faced with a change to uniform skill-demand. Future work will include finding a method that is capable of adapting to a wider variety of skill congregation environments, making dealing with skills congregation found in real-world problems more successful.

References

[1] S. Abdallah and V. Lesser. Multiagent reinforcement learning and self-organization in a network of agents. In *AAMAS*, 2007, 72–179.

[2] L. Barton and V. H. Allan. Information Sharing in an agent-organized network, intelligent agent technology. In *IAT,* 2007, 89–92.

[3] L. Barton and V. H. Allan. Methods for coalition formation in adaptation-based social networks. In *CIA*, 2007, 285–297.

[4] B. Bulka, M. Gaston, and M. desJardins. Local strategy learning in networked multi-agent team formation. In *AAMAS*, 2007, 29–45.

[5] A. M. Easwaran and J. Pitt. Supply chain formation in open, market-based multi-agent systems. *Int. J. .Computational Intell. and Applic.,* 2(3): 349–363, 2002.

[6] M. E. Gaston and M. des Jardins. Agent-organized networks for dynamic team formation. In *AAMAS*, 2005, 230–237.

[7] S. Kraus, O. Shehory, and G. Taase. Coalition formation with uncertain heterogeneous information. In *AAMAS*, 2003, 1–8.

[8] K. Lerman and O. Shehory. Coalition formation for large-scale electronic markets. In *4th Int. Conf. on Multi-Agent Systems*, July 2000, 167–174.

[9] E. Manisterski, E. David, S. Kraus, and N. R. Jennings. Forming efficient agent groups for completing complex tasks. In *AAMAS*, 2006, 834–841.

[10] P. V. Sander, D. Peleshchuk, and B. J. Grosz. A scalable, distributed algorithm for efficient task allocation. In *AAMAS*, 2002, 1191–1198.

[11] O. Shehory. A scalable agent location mechanism. In *Workshop on Intell. Agents VI, Agent Theories, Architectures, and Languages: Lecture Notes In Computer Science*, vol. 1757, 1999, 162–172.

[12] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intell.,* 101:165–200, May 1998.

[13] T. Sugawara, S. Kurihara, T. Hirotsu, K. Fukuda, S. Sato, and O. Akashi. Total performance by local agent selection strategies in multi-agent systems. In *AAMAS*, 2006, 601–608.

[14] W. E. Walsh and M. P. Wellman. Modeling supply chain formation in multiagent systems. In *Agent Mediated Electronic Commerce II; Lecture Notes in Computer Science*, vol.1788, 2000, 94–101.

[15] M. d. Weerdt, Y. Zhang, and T. Klos. Distributed task allocation in social networks. In *AAMAS*, 2007.

CHAPTER 5

CONCLUSION

We have added to the knowledge base of rewiring strategies in agent-organized networks, showing how agents can improve their coalition formation abilities by making decisions on whom to connect to in the network, for which tasks to form coalitions, and how long to wait for the coalition to complete. We have shown how strategic task selection and task impatience improve performance when applied alone and improve performance even more when together.

We have also shown how different agent types compare when more information is available when choosing to rewire an agent's network connections. We have shown that inventory rewiring compares well to structural rewiring until the agents approach complete information sharing.

We have also shown inventory rewiring to outperform other agent rewiring strategies when skill-demand is non-uniform and constant. We have also shown inventory agents to outperform other agent rewiring strategies with congregate skill-supply and changing skill-demand.

CHAPTER 6

FUTURE WORK

Future work should examine the possibility of mixing agent types to improve performance even more. It may be possible to merge the best attributes of two agent types into one agent superior to either individual agent. Another possibility would be having both agent types represented in the agent population, potentially achieving the same result.

Another area of future research would be agents who adapt to changing task loads. We have examined increased task loads, but the rate at which tasks were created was constant for the entire test. Agents who adapt to changing task loads would be better suited to real world scenarios, aiding the fielding of agent technology in a variety of applications.

Finally, research could focus on adapting to changing skill-demand. We have shown that certain agent types perform well for specific skill-supply and skill-demand scenarios. However, in real world applications, the skill-demand may not be constant, and agents who could excel in a majority of skill-supply and skill-demand settings would be more useful.

APPENDICES

APPENDIX A

TEST SETTINGS INPUT FILE

```
<AON AgentCount="100" MaxInitialDistance="100" MaxIterations="2000"
MinAdaptTimeStep="1000"
 MaxActiveTime="10" MaxAdvertisedTime="10" MaxCommittedTime="10"
MaxNodeDegree="20"
 CommunicationDepth="2" NeighborhoodRadius="10" UseCommandSkills="false"
 NumberOfSkills="10" SkillsPerTask="10" SkillsPerAgent="1"
TaskAnnounceTime="10" WriteAgentData="false"
 UpdateInterval="1000" PaintInterval="1000" ScreenShotInterval="500"
SaveScreenShot="false"
 DrawAgentIDs="true" DrawTaskIDs="true" DrawAON="false" WriteDataInterval="100"
 OutputFilename="AONResults" ReportDirectory="AON28_Test"
ReportDescription="Degree Limit 20">
  <Tests>
    <Test1 Name="Diversity" Adaptation="Diversity" OutputDirectory="Diversity"
Description="Diversity" ChangeAgentRatio="false"/>
    <Test2 Name="Egalitarian" Adaptation="Egalitarian"
OutputDirectory="Egalitarian" Description="Egalitarian"
ChangeAgentRatio="false"/>
    <Test3 Name="Inventory" Adaptation="Inventory" OutputDirectory="Inventory"
Description="Inventory" ChangeAgentRatio="false"/>
    <Test4 Name="Performance" Adaptation="Performance"
OutputDirectory="Performance" Description="Performance"
ChangeAgentRatio="false"/>
    <Test5 Name="Structural" Adaptation="Structural"
OutputDirectory="Structural" Description="Structural"
ChangeAgentRatio="false"/>
    <Test6 Name="Uniform" Adaptation="Uniform" OutputDirectory="Uniform"
Description="Uniform" ChangeAgentRatio="false"/>
  </Tests>
  <AgentTypes>
    <AgentType1 Name="Diversity" Ratio="0.1" Color="pink">
    </AgentType1>
    <AgentType2 Name="Egalitarian" Ratio="0.2" Color="magenta">
    </AgentType2>
    <AgentType3 Name="Inventory" Ratio="0.3" Color="cyan">
    </AgentType3>
    <AgentType4 Name="Performance" Ratio="1.0" Color="orange">
    </AgentType4>
    <AgentType5 Name="Structural" Ratio="1.0" Color="green">
    </AgentType5>
    <AgentType6 Name="Uniform" Ratio="1.0" Color="yellow">
    </AgentType6>
  </AgentTypes>
  <Agents Filename="AON_Agents.xml"/>
</AON>
```

APPENDIX B

SOURCE CODE

```
--------------------------------AgentList.java-----------------------------------
-

import java.util.*;

/**
 * XMLElement Class
 *   The Agent List class is used to manage the list of agents in the system.
 *
 */
public class AgentList{
   private Vector<AgentX> m_agents;
                              /* The vector storing the agent objects
*/

   /* Default constructor
*/
   public AgentList(){
      m_agents = new Vector();
   }//constructor

   /* Copy constructor
*/
   public AgentList(Vector agents){
      m_agents = agents;
   }//constructor

   /* This function adds the specified agent object to the list
*/
   public void add(AgentX agent){
      m_agents.add(agent);
   }//add

   /* This function returns the agent matched with the specified  agent id
*/
   /*  or returns null if no such agent is found
*/
   public AgentX getAgent(int id){
      for(int i = 0; i < m_agents.size(); i++){
         AgentX agent = m_agents.elementAt(i);
         if(agent.getID() == id){
            return agent;
         }//if
      }//for
      return null;
   }//getAgent

   /* This function returns the agent matched with the specified index in the
*/
   /*  agent list, or null if the specified index is invalid
*/
   public AgentX getAgentByIndex(int index){
      AgentX agentFound = null;
      if(index >= 0 && index < m_agents.size()){
         agentFound = m_agents.elementAt(index);
      }//if
      return agentFound;
   }//getAgent
```

```
   /* This function returns true if the specified agent id is found in the
*/
   /*  agent list, or false if the agent id does not exist in the list
*/
   public boolean containsAgent(int id){
      for(int i = 0; i < m_agents.size(); i++){
         AgentX agent = m_agents.elementAt(i);
         if(agent.getID() == id){
            return true;
         }//if
      }//for
      return false;
   }//containsAgent

   /* This function returns the number of agents stored in the agent list
*/
   public int size(){
      return m_agents.size();
   }//size
}//class AgentList

--------------------------------AgentList.java----------------------------------
-

-------------------------AgentRecommendation.java-----------------------------
-

/**
 * AgentRecommendation Class
 *   This class is used to pass an agent recommendation between agents.
 *   The agent being recommended is identified by m_id and the recommendation
 *   is quantified by m_value
 */
public class AgentRecommendation{
   private int m_id;         /* The id of the agent being recommended
*/
   private float m_value;    /* The value the agent has been assigned
*/

   /* Default constructor
*/
   public AgentRecommendation(){
      m_id = 0;
      m_value = 0.0f;
   }//AgentRecommendation

   /* Constructor passing initial recommendation values
*/
   public AgentRecommendation(int id, float value){
      m_id = id;
      m_value = value;
   }//AgentRecommendation

   /* Constructor passing initial recommendation values
*/
   public AgentRecommendation(int id, int value){
      m_id = id;
      m_value = (float)value;
   }//AgentRecommendation

   /* The function returns the agent id for the agent being recommended
*/
```

```
   public int getID(){
      return m_id;
   }//getID

   /* This function returns the value of the agent being recommended
*/
   public float getValue(){
      return m_value;
   }//getValue

   /* This function returns a string representation
*/
   /*  of the agent recommendation
*/
   public String toString(){
      return "Agent ID=" + m_id + " Value=" + m_value;
   }//toString
}//AgentRecommendation
```

----------AgentRecommendation.java----------
-

----------AgentX.java----------
-

```
import java.awt.*;
import java.util.*;
import java.io.IOException;

/**
 * AgentX Class
 *   This is the base agent class which defines common agent structure such as
 *   the update funcion, the rewire function, etc.
 *   Other agent classes extend the AgentX class, overidding base functions
 *   to fit specific functionality to each agent within the overall framework
 *   defined here in AgentX
 */
public class AgentX{
   /* These constants represent the possible values of m_state
*/
   final int C_UNCOMMITTED = 0,
                             /* The agent is not committed to a task
*/
             C_COMMITTED = 1,
                             /* The agent is committed to a task
*/
             C_ACTIVE = 2;   /* The agent is committed and working on a task
*/

   /* These constants represent the possible values of m_lastWorkCatagory
*/
   final int C_WAITED = 0,   /* Last agent action was to wait
*/
             C_REWIRED = 1,  /* Last agent action was to rewire
*/
```

```
            C_PROPOSED = 2, /* Last agent action was to propose a task
*/
            C_JOINED = 3,   /* Last agent action was to join a task
*/
            C_WORKED = 4,   /* Last agent action was to work on a task
*/
            C_TRAVELLED = 5;
                            /* Last agent action was to travel
*/

   int m_id;                /* The agent ID
*/
   int m_taskID;            /* The task ID of the task agent is working on
*/
   int m_x;                 /* The x pixel location of the agent
*/
   int m_y;                 /* The y pixel location of the agent
*/
   String m_type;           /* The agent's agent type
*/
   int m_springX;           /* The agent's spring adjusted x pixel location
*/
   int m_springY;           /* The agent's spring adjusted y pixel location
*/
   int m_state;             /* The current state of the agent
*/
   int m_adaptationRate;    /* The adaptation rate used for when to change
*/
                            /*   the agent's neighbor connections
*/
   int m_activeTimeRemaining;/* The amount of time the agent will be active
*/
   int m_committedTimeRemaining;
                            /* The amount of time the agent will be
*/
                            /*   committed to a task
*/
   int m_maxActiveTime;     /* The maximum amount of time the agent will be
*/
                            /*   actively working on a task
*/
   int m_maxCommittedTime;  /* The maximum amount of time the agent will be
*/
                            /*   committed to a task
*/
   float m_waitPercent;     /* The percentage used for choosing when to wait
*/
   int m_edgesChanged;      /* The number of edges to neighbors changed in
*/
                            /*   the current agent update cycle
*/
   NeighborList m_neighbors; /* The list of agents who the agent is connected
*/
                            /*   to in the network
*/
   int m_skill;             /* The skill the agent possesses
*/
   int m_systemSkillCount;  /* The total number of skills that agents can
*/
                            /*   possess and tasks can require
*/
   AgentList m_agents;      /* The list of all the agents in the network
*/
```

```
                              /*   (even those not directly connected to the
*/
                              /*    current agent)
*/
   Color m_color;             /* The color used to draw the agent's oval
*/
   String m_teamString;       /* The unique identifying string for the group
*/
                              /*   of agents working on a task
*/
   boolean m_adaptNetwork;    /* A flag determining whether the agent can
*/
                              /*   change network connections
*/
   AONTask m_agentTask;       /* The task the agent is committed to
*/
   int m_taskRange;           /* This specifies the maximum distance an agent
*/
                              /*   can be from a task while working on the
*/
                              /*   task
*/
   int m_maxDegree;           /* The maximum number of neighbors the agent can
*/
                              /*   have
*/
   int m_communicationDepth; /* The depth in the network an agent can look
*/
                              /*   when gathering information to rewire
*/
   int m_neighborhoodRadius; /* The depth in the network an agent can look
*/
                              /*   when counting neighborhood skills
*/
   int m_lastWorkCatagory;    /* The work status for the previous update cycle
*/
   TestAON parent;            /* A reference to the TestAON object
*/
   Random rand;               /* A global random class object
*/
   Vector<Point> m_plan;      /* The plan for travelling along roads specified
*/
                              /*   by the Roads object m_roads
*/
   boolean m_useRoads;        /* A flag determining whether to travel along
*/
                              /*   roads specified by the Roads object m_roads
*/
   Roads m_roads;             /* A class defining roads the agent travels on
*/

   String m_debugString;      /* Contains debug information about agent state
*/

   int m_timeWaiting;         /* The amount of time waiting
*/
   int m_timeActive;          /* The amount of time active
*/
   int m_totalTime;           /* The total amount of time
*/

   int m_numberOfTeamsJoined;/* The number of teams the agent has joined
*/
```

```
   int m_numberOfSuccessfulTeamsJoined;
                              /* The number of successful teams joined
*/

   int m_taskDistance;        /* The distance from the agent to agent's task
*/
   boolean m_moveToTask;      /* Flag for whether the agent navigates to tasks
*/
                              /*   or not
*/
   CSVFile m_agentData;       /* Comma Seperated File used to save agent data
*/

   int m_lastX;               /* The agent's previous x pixel position
*/
   int m_lastY;               /* The agent's previous y pixel position
*/
   int m_lastTaskDistance;    /* The previous distance from the agent to the
*/
                              /*   agent's task
*/
   int m_shapeID;             /* The id of the shape the agent is currently
*/
                              /*   located at in the roads
*/
   int m_distanceTravelled;   /* The distance the agent has travelled
*/
   int m_maximumPlanTime;     /* The maximum amount of time the agent is
*/
                              /*   allowed to plan its travels in roads
*/
   NeighborhoodStats m_neighborhoodStats;
                              /* The statistics for the agent's neighborhood
*/
   int m_neighborhoodStatsUpdateCount;
                              /* A count of timesteps since the last update
*/
                              /*   of the neighborhood statistics
*/
   boolean m_rewired;         /* True when the agent has rewired since the
*/
                              /*   last update of neighborhood statistics
*/
   int m_joinedImpossibleTaskCount;
                              /* The number of impossible tasks joined
*/
   int m_proposedImpossibleTaskCount;
                              /* The number of impossible tasks proposed
*/
   Vector m_teamsJoinedHistory;
                              /* List of task id's for tasks agent has joined
*/
   int m_repeatTaskCount;     /* The number of times agent commits to a task
*/
                              /*   that it has committed to in the past
*/
   float m_neighborhoodSkillDistance;
                              /* The average number of network connections to
*/
                              /*   each skill when computing neighborhood
*/
                              /*   statistics
*/
```

```
   int m_neighborhoodSkillShortage;
                              /* The number of skills that were not found when
*/
                              /*   computing neighborhood statistics
*/
   Vector<Integer> m_unfilledSkillsCount;
                              /* The number of times each skill has not been
*/
                              /*   filled in the tasks committed to so far
*/
   SkillShortageHistory m_unfilledSkillHistory;
                              /* The history of how many times each skill has
*/
   int m_skillHistoryLength; /*   gone unfilled in tasks committed to within
*/
                              /*   the recent skill history
*/

   /* Default constructor
*/
   public AgentX(){
      m_neighbors = new NeighborList();
      m_agents = new AgentList();
      m_plan = new Vector();
      rand = new Random(System.currentTimeMillis()*m_id);
      m_shapeID = -1;
      m_maximumPlanTime = 10000;
      m_teamsJoinedHistory = new Vector();
      m_skillHistoryLength = -1;
   }//constructor

   /* constructor
*/
   public AgentX(int id){
      m_id = id;
      m_neighbors = new NeighborList();
      m_agents = new AgentList();
      m_plan = new Vector();
      rand = new Random(System.currentTimeMillis()*m_id);
      m_shapeID = -1;
      m_maximumPlanTime = 10000;
      m_teamsJoinedHistory = new Vector();
      m_skillHistoryLength = -1;
   }//constructor

   /* constructor
*/
   public AgentX(TestAON p,int id, int x, int y, String type, Vector agents){
      m_id = id;
      parent = p;
      m_taskID = -1;
      m_x = x;
      m_y = y;
      m_springX = x;
      m_springY = y;
      m_type = type;
      m_state = C_UNCOMMITTED;
      m_adaptationRate = 1;
      m_activeTimeRemaining = 0;
      m_committedTimeRemaining = 0;
      m_neighbors = new NeighborList();
      m_skill = -1;
```

```
      m_agents = new AgentList(agents);
      m_teamString = "";
      m_adaptNetwork = true;
      m_plan = new Vector();

      m_moveToTask = false;

      m_debugString = "";
      rand = new Random(System.currentTimeMillis()*m_id);
      m_agentData = new CSVFile();
      m_numberOfTeamsJoined =0;
      m_numberOfSuccessfulTeamsJoined=0;
      m_shapeID = -1;
      m_maximumPlanTime = 10000;
      m_teamsJoinedHistory = new Vector();
      m_skillHistoryLength = -1;
   }//constructor

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   /* This function sets the value for the maximum time the agent can remain
*/
   /*   actively working on a task
*/
   public void setMaxActiveTime(int maxActiveTime){
      m_maxActiveTime = maxActiveTime;
   }//setMaxActiveTime

   /* This function sets the value for the maximum time the agent can remain
*/
```

```
   /*   committed to a task
*/
   public void setMaxCommittedTime(int maxCommittedTime){
      m_maxCommittedTime = maxCommittedTime;
   }//setMaxCommittedTime

   /* This function sets the maximum number of connections the agent can have
*/
   /*   to other agents
*/
   public void setMaxNodeDegree(int maxNodeDegree){
      m_maxDegree = maxNodeDegree;
   }//setMaxNodeDegree

   /* This function sets the depth in the network the agent looks when
*/
   /*   information for rewiring its local network connections
*/
   public void setCommunicationDepth(int communicationDepth){
      m_communicationDepth = communicationDepth;
   }//setCommunicationDepth

   /* This function sets the depth in the network the agent looks when
*/
   /*   counting neighborhood skills
*/
   public void setNeighborhoodRadius(int neighborhoodRadius){
      m_neighborhoodRadius = neighborhoodRadius;
   }//setNeighborhoodRadius

   /* This function sets the maximum distance the agent can be from a task
*/
   /*   and be actively working on the task
*/
   public void setTaskRange(int taskRange){
      m_taskRange = taskRange;
   }//setTaskRange

   /* This function sets the value for the rate the agent adapts its
*/
   /*   connections to agents in the network
*/
   public void setAdaptationRate(int rate){
      m_adaptationRate = rate;
   }//setAdaptationRate

   /* This function sets the flag for whether the agent moves to tasks
*/
   /*   to be within range to work on the task
*/
   public void setMoveToTask(boolean value){
      m_moveToTask = value;
   }//setMoveToTask

   /* This function sets the value of the random seed, this random seed is
*/
   /*   then used to create a global random object
*/
   public void setRandomSeed(int randomSeed){
      rand = new Random(randomSeed);
   }//setRandomSeed
```

```
   /* This function sets the roads file for the agent to navigate on
*/
   public void setRoads(Roads roads){
      if(m_shapeID < 0){
         m_shapeID = 1;
      }//if
      m_useRoads = true;
      m_roads = roads;
   }//setRoads

   /* This function sets the number of skills available in the system
*/
   public void setSystemSkillsCount(int count){
      m_systemSkillCount = count;
   }//setSystemSkillsCount

   /* This function returns the number of times the agent has joined an
*/
   /*   impossible task
*/
   public int getJoinedImpossibleCount(){
      return m_joinedImpossibleTaskCount;
   }//getJoinedImpossibleCount

   /* This function returns the number of times the agent has proposed an
*/
   /*   impossible task
*/
   public int getProposedImpossibleCount(){
      return m_proposedImpossibleTaskCount;
   }//getProposedImpossibleCount

   /* This function returns the number of times the agent commits to a task
*/
   /*   it was committed to in the past
*/
   public int getRepeatTaskCount(){
      return m_repeatTaskCount;
   }//getRepeatTaskCount

   /* This function returns the average number of network connections to each
*/
   /*   kind of skill in the local neighborhood
*/
   public float getNeighborhoodSkillDistance(){
      return m_neighborhoodSkillDistance;
   }//getNeighborhoodSkillDistance

   /* This function returns the number of skills not found when counting
*/
   /*   neighborhood skills
*/
   public int getNeighborhoodSkillShortage(){
      return m_neighborhoodSkillShortage;
   }//getNeighborhoodSkillShortage

   /* This function returns the length of the agent's skill history
*/
   public int getSkillHistoryLength(){
      return m_skillHistoryLength;
   }//getSkillHistoryLength
```

```
   /* This function sets the length of the agent's skill history
*/
   public void setSkillHistoryLength(int skillHistoryLength){
      m_skillHistoryLength = skillHistoryLength;
   }//setSkillHistoryLength

   /* This function writes the specified string to the parent TestAON log
*/
   public void writeLog(String line){
      if(parent != null){
         parent.writeLog(line);
      }//if
   }//writeLog

   /* This function writes the data for the agent in the comma seperated file
*/
   /*   that coincides with the specified system iteration and performance
*/
   public void writeAgentData(int iteration, float performance){
      // If the comma seperated file is empty
      if(m_agentData.getLineCount() == 0){
         // Write header names in the first row of the  comma seperated file
         String headerValues[] = {"iteration", "agent_performance",
                                  "neighbor_performance", "system_performance",
                                  "density", "percent_active",
"percent_waiting",
                                  "degree", "joined_impossible",
                                  "proposed_impossible", "repeat_task",
                                  "neighborhood_skill_distance", "neighbors"};
         m_agentData.addRow(headerValues);
      }//else

      float neighborPerformance = getAverageNeighborPerformance();
      String agentValues[] = {String.valueOf(iteration),
String.valueOf(getPerformance()),
                              String.valueOf(neighborPerformance),
String.valueOf(performance),
                              String.valueOf(getDensity(3)),
String.valueOf(getPercentActive()), String.valueOf(getPercentWaiting()),
                              String.valueOf(getDegree()),
String.valueOf(m_joinedImpossibleTaskCount),
                              String.valueOf(m_proposedImpossibleTaskCount),
String.valueOf(m_repeatTaskCount),
                              String.valueOf(m_neighborhoodSkillDistance),
getNeighborsString()};
      m_agentData.addRow(agentValues);
   }//writeAgentData

   /* This function saves the comma seperated file containing the agent's
*/
   /*   information for each time step to the specified file name
*/
   public void saveAgentData(String filename){
      m_agentData.save(filename);
   }//saveAgentData

   /* This function returns the agent's id number
*/
   public int getID(){
      return m_id;
   }//getID
```

```
   /* This function returns the agent's id and skill id in the string format
*/
   /*   'id/skill_id'
*/
   public String getStringID(){
      return m_id + "/" + m_skill;
   }//getID

   /* This function returns the id of the task the agent is committed to
*/
   public int getTaskID(){
      return m_taskID;
   }//getTaskID

   /* This function sets the id of the task the agent is committed to
*/
   public void setTaskID(int taskID){
      m_taskID = taskID;
   }//setTaskID

   /* This function returns the agent's x pixel location
*/
   public int getX(){
      return m_x;
   }//getX

   /* This function sets the agent's x pixel location
*/
   public void setX(int x){
      m_x = x;
   }//setX

   /* This function returns the agent's y pixel location
*/
   public int getY(){
      return m_y;
   }//getY

   /* This function sets the agent's y pixel location
*/
   public void setY(int y){
      m_y = y;
   }//setY

   /* This function returns the agent's spring adjusted x pixel location
*/
   public int getSpringX(){
      return m_springX;
   }//getSpringX

   /* This function returns the agent's spring adjusted y pixel location
*/
   public int getSpringY(){
      return m_springY;
   }//getSpringY

   /* This function returns the agent's type description string
*/
   public String getType(){
      return m_type;
   }//getType
```

```
   /* This function returns the agent's state
*/
   public int getState(){
      return m_state;
   }//getState

   /* This function sets the agent's state
*/
   public void setState(int state){
      m_state = state;
      if(m_state == C_COMMITTED){
         m_committedTimeRemaining = m_maxCommittedTime;
      }else if(m_state == C_ACTIVE){
         m_activeTimeRemaining = m_agentTask.getDuration();
      }//else if
   }//setState

   /* This function returns the agent's work catagory
*/
   /*
*/
   public int getWorkCatagory(){
      return m_lastWorkCatagory;
   }//getWorkCatagory

   /* This function returns the total distance travelled by the agent
*/
   public int getDistanceTravelled(){
      int distance = m_distanceTravelled;
      return distance;
   }//getDistanceTravelled

   /* This function returns the color to draw the agent's oval
*/
   public Color getColor(){
      switch(m_state){
         case(C_UNCOMMITTED):
            return Color.blue;
         case(C_COMMITTED):
            return Color.green;
         case(C_ACTIVE):
            return Color.red;
         default:
      }//switch
      return m_color;
   }//getColor

   /* This function sets the color of the agent
*/
   public void setColor(Color c){
      m_color = c;
   }//setColor

   /* This function returns the distance from the agent to the task the agent
*/
   /*   is committed to
*/
   public int getTaskDistance(){
      int taskX = m_agentTask.getX(), taskY = m_agentTask.getY();
      return (int)Math.sqrt((m_x-taskX)*(m_x-taskX) + (m_y-taskY)*(m_y-taskY));
   }//getTaskDistance
```

```
   /* This function recomputes the distance from the agent to the task the
*/
   /*   agent is committed to and stores this distance
*/
   public void updateTaskDistance(){
      m_taskDistance = getTaskDistance();
   }//updateTaskDistance

   /* This function sets the maximum number of connections the agent can have
*/
   /*   to neighboring agents
*/
   public void setMaxDegree(int maxDegree){
      m_maxDegree = maxDegree;
   }//setMaxDegree

   /* This function returns whether the agent is available to commit to a
*/
   /*   task
*/
   public boolean isAvailable(){
      return m_taskID < 0;
   }//isAvailable

   /* This function returns the color of the connection between the agent and
*/
   /*   another agent with the specified agent id
*/
   public Color getNeighborColor(int neighborID){
      Color color = Color.black;
      int life = m_neighbors.getItem(neighborID).connectionLife();
      if(life > 10){
         color = Color.gray;
      }else if(life > 5){
         color = Color.orange;
      }else{
         color = Color.red;
      }//else
      return color;
   }//getNeighborColor

   /* This function returns the number of neighbor connections the agent has
*/
   /*   changed by rewiring
*/
   public int getChangedEdgesCount(){
      return m_edgesChanged;
   }//getChangedEdgesCount

   /* This function returns a string listing the agent id and neighbor rating
*/
   /*   of each neighbor the agent has
*/
   public String getNeighborsString(){
      String result = "";
      for(int i = 0; i < m_neighbors.getLength(); i++){
         NeighborLink neighbor = m_neighbors.getItem(i);
         float neighborRating = neighbor.agentRating();
         int neighborID = neighbor.agentID();
         result = result + " " + neighborID + ":" + neighborRating;
      }//for
      return result;
   }//getNeighborsString
```

```
   /* This function returns a string representation of the agent's skill
*/
   public String getSkillsString(){
      return " " + m_skill;
   }//getSkillsString

   /* This function returns the number of agents within the specified number
*/
   /*   of network connections of the agent
*/
   public int getDensity(int level){
      if(level == 0){
         return 1;
      }else{
         Vector agentsTouched = new Vector();
         agentsTouched.add(new Integer(m_id));
         for(int i = 0; i < m_neighbors.getLength(); i++){
            Integer agentID = (Integer)m_neighbors.getItem(i).agentID();
            if(!contains(agentID,agentsTouched)){
               AgentX agent = getAgent(agentID.intValue());
               if(agent != null){
                  agent.getDensity(level-1,agentsTouched);
               }//if
            }//if
         }//for
         return getLength(agentsTouched);
      }//else
   }//getDensity

   /* This function helps compute neighborhood density by adding neighbors
*/
   /*   to the agents touched list until the level variable reaches 0
*/
   public void getDensity(int level, Vector agentsTouched){
      agentsTouched.add(new Integer(m_id));
      if(level > 0){
         for(int i = 0; i < m_neighbors.getLength(); i++){
            Integer agentID = (Integer)m_neighbors.getItem(i).agentID();
            if(!contains(agentID,agentsTouched)){
               AgentX agent = getAgent(agentID.intValue());
               if(agent != null){
                  agent.getDensity(level-1,agentsTouched);
               }//if
            }//if
         }//for
      }//if
   }//getDensity

   /* This function returns the number of network connections the agent has
*/
   /*   to other agents in the network
*/
   public int getDegree(){
      return m_neighbors.getLength();
   }//getDegree

   /* This function returns the list of neighboring agents for this agent
*/
   public NeighborList getNeighbors(){
      return m_neighbors;
   }//getNeighbors
```

```
   /* This function returns true if the specified agent id identifies one of
*/
   /*   the agents this agent is connected to by network connections
*/
   public boolean hasNeighbor(int agentID){
      return m_neighbors.containsAgent(agentID);
   }//hasNeighbor

   /* This function returns the skill the agent possesses
*/
   public int getSkills(){
      return m_skill;
   }//getSkills

   /* This function returns the string that identifies the team for the task
*/
   /*   the agent is committed to
*/
   public String getTeamString(){
      return m_teamString;
   }//getTeamString

   /* This function sets the flag that determines whether the agent changes
*/
   /*   its network connections to its neighbors
*/
   public void setAdaptNetwork(boolean adapt){
      m_adaptNetwork = adapt;
   }//setAdaptNetwork

   /* This function returns the agent with the specified id in the agent list
*/
   public AgentX getAgent(int id){
      return m_agents.getAgent(id);
   }//getAgent

   /* This function returns the number of agents in the agent list
*/
   public int getAgentCount(){
      return m_agents.size();
   }//getAgentCount

   /* This function returns the percentage of successful teams this agent has
*/
   /*   joined
*/
   public float getPerformance(){
      if(m_numberOfTeamsJoined == 0){
         return 0.0f;
      }else{
         return (float)m_numberOfSuccessfulTeamsJoined / m_numberOfTeamsJoined;
      }//else
   }//getPerformance

   /* This function returns the average performance estimate for the agent's
*/
   /*   neighbors
*/
   public float getAverageNeighborPerformance(){
      float sum = 0.0f;
      if(m_neighbors.getLength() == 0){
         return 0.0f;
      }//if
```

```
      for(int i = 0; i < m_neighbors.getLength(); i++){
         int id = m_neighbors.getItem(i).agentID();
         AgentX agent = (AgentX)getAgent(id);
         sum = sum + agent.getPerformance();
      }//for
      return (float)sum / m_neighbors.getLength();
   }//getAverageNeighborPerformance

   /* This function returns the percentage of the time the agent has been
*/
   /*   actively working on tasks
*/
   public float getPercentActive(){
      if(m_totalTime == 0){
         return 0.0f;
      }else{
         return m_timeActive / m_totalTime;
      }//else
   }//getPercentActive

   /* This function returns the percentage of the time the agent has been
*/
   /*   waiting
*/
   public float getPercentWaiting(){
      if(m_totalTime == 0){
         return 0.0f;
      }else{
         return m_timeWaiting / m_totalTime;
      }//else
   }//getPercentWaiting

   /* This function returns the shape id for the shape in the roads object
*/
   /*   the agent is located at in the m_roads object
*/
   public int getShapeID(){
      return m_shapeID;
   }//getShapeID

   /* This function sets the shape id for the shape the agent is located at
*/
   /*   in the roads object m_roads
*/
   public void setShapeID(int id){
      m_shapeID = id;
   }//setShapeID

   /* This function notifies the agent specified by the id that it has been
*/
   /*   added as a neighbor to this agent
*/
   public void sendAddNeighborMessage(int id){
      AgentX newNeighbor = getAgent(id);
      if(newNeighbor != null){
         newNeighbor.notifyAddNeighbor(m_id);
      }//if
   }//sendAddNeighborMessage

   /* This function adds the agent specified by the id as a neighbor to this
*/
   /*   agent
*/
```

```
   public boolean addNeighbor(int id){
      AgentX neighbor = getAgent(id);
      if(m_neighbors.getLength() > m_maxDegree ||
         neighbor.getDegree() >= m_maxDegree){
         return false;
      }//if
      if(m_neighbors.containsAgent(id) || id == m_id){
         return false;
      }//if
      m_neighbors.addAgent(id);
      sendAddNeighborMessage(id);

      return true;
   }//addNeighbor


   /* This function notifies the agent specified by the id that it has been
*/
   /*   removed as a neighbor to this agent
*/
   public void sendRemoveNeighborMessage(int id){
      AgentX oldNeighbor = getAgent(id);
      if(oldNeighbor != null){
         oldNeighbor.notifyRemoveNeighbor(m_id);
      }//if
   }//sendRemoveNeighborMessage

   /* This function removes the agent specified by the id as a neighbor of
*/
   /*   this agent
*/
   public void removeNeighbor(int id){
      if(m_neighbors.containsAgent(id)){
         m_neighbors.removeAgent(id);
         sendRemoveNeighborMessage(id);
      }//if
   }//removeNeighbor

   /* This function adds the agent specified by id as a neighbor to this
*/
   /*   agent without notifying the added neighbor
*/
   public void addNeighborNoNotify(int id){
      if(!m_neighbors.containsAgent(id) && id != m_id){
         m_neighbors.addAgent(id);
      }//if
   }//addNeighborNoNotify

   /* This function removes the agent specified by id as a neighbor to this
*/
   /*   agent without notifying the removed neighbor
*/
   protected void removeNeighborNoNotify(int id){
      if(m_neighbors.containsAgent(id)){
         m_neighbors.removeAgent(id);
      }//if
   }//removeNeighbor

   /* This function is used by other agents to tell this agent that it has
*/
   /*   been added as a neighbor
*/
   public void notifyAddNeighbor(int id){
```

```
      if(m_neighbors.getLength() >= m_maxDegree){
         sendRemoveNeighborMessage(m_id);
         return;
      }//if
      addNeighborNoNotify(id);
   }//notifyAddNeighbor

   /* This function is used by other agents to tell this agent that it has
*/
   /*   been removed as a neighbor
*/
   public void notifyRemoveNeighbor(int id){
      removeNeighborNoNotify(id);
   }//notifyRemoveNeighbor

   /* This function removes all of the agent's neigbhors
*/
   public void removeAllNeighbors(){
      int length = m_neighbors.getLength();
      for(int i = 0; i < length; i++){
         int id = m_neighbors.getItem(0).agentID();
         removeNeighbor(id);
      }//for
   }//removeAllNeighbors

   /* This function returns the task the agent is currently committed to
*/
   public AONTask getTask(){
      return m_agentTask;
   }//getTask

   /* This function pauses execution until the user presses enter at the
*/
   /*   console (used for debugging purposes)
*/
   public void pauseTest(String message){
      try{
         char input = ' ';
         byte temp[] = new byte[1];
         System.out.print("\n" + message + "\nPress <enter> to continue...");
         while(input != '\n'){
            System.in.read(temp);
            input = (char)temp[0];
         }//while
      }catch(IOException ex){
      }//catch
   }//pauseTest

   /* This function sets the skill possessed by the agent
*/
   public void addSkill(int skill){
         m_skill = skill;
   }//addSkill

   /* This function is called to update the agent for the current time step
*/
   /*   The list of system tasks is passed in as an AONTaskList
*/
   public void update(AONTaskList tasks){
      if(m_lastWorkCatagory != C_PROPOSED){
         m_lastWorkCatagory = C_WAITED;
      }//if
```

```
      updateNeighborhoodStats();

      initializeUnfilledSkillCount();

      //System.out.println("update:");
      writeLog(m_id + " update AgentX:");
      int taskDistance = 0;

      switch(m_state){
         case(C_UNCOMMITTED):
            m_debugString = "UNCOMMITTED";
            updateUncommitted(tasks);
            if(m_state == C_COMMITTED){
               updateTaskDistance();
            }//if
            break;
         case(C_COMMITTED):
            m_debugString = "COMMITTED";
            updateCommitted(tasks);
            break;
         case(C_ACTIVE):
            m_debugString = "ACTIVE";

            AONTask task = (AONTask)tasks.getItem(m_taskID);
            taskDistance = getDistance(m_x,m_y,task.getX(),task.getY());
            boolean inRange = (taskDistance < m_taskRange);
            if(inRange){
               writeLog(m_id + " Task Message: Task in range! Active Time
remaining=" + m_activeTimeRemaining + " taskDistance=" + taskDistance + "
m_taskID=" + m_taskID + " m_lastTaskDistance=" + m_lastTaskDistance + task);
            }else{
               String otherStuff = "";
               if(m_x != m_lastX || m_y != m_lastY)otherStuff = "Agent moved:
x,y=" + m_x + "," + m_y + " lastX,lastY=" + m_lastX + "," + m_lastY;
               writeLog(m_id + " Task Message: OUT OF RANGE OF ACTIVE TASK!
Active Time remaining=" + m_activeTimeRemaining + " taskDistance=" +
taskDistance + " m_taskID=" + m_taskID + " m_lastTaskDistance=" +
m_lastTaskDistance + task + otherStuff);
            }//else

            updateActive();
            break;
         default:
      }//switch

      m_lastX = m_x;
      m_lastY = m_y;
      m_lastTaskDistance = taskDistance;
      incrementNeighborConnections();
      updateNeighborRatings();
   }//update

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
      writeLog(m_id + " updateUncommitted:");
      m_edgesChanged = 0;
      if(chooseToAdapt()){
         rewire();
      }else{
         if(joinTeam(tasks)){
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
```

```
         }//if
      }//else
   }//updateUncommitted

   /* This function returns a random neighbor agent id as a neighbor to be
*/
   /*   removed
*/
   public int pickRemoveNeighbor(){
      int neighborIndex = rand.nextInt(m_neighbors.getLength()-1);
      return m_neighbors.getItem(neighborIndex).agentID();
   }//pickRemoveNeighbor

   /* This function returns the distance from x1,y1 to x2,y2
*/
   public int getDistance(int x1, int y1, int x2, int y2){
      return (int)Math.sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
   }//getDistance

   /* This function updates the agent when its state is committed
*/
   public void updateCommitted(AONTaskList tasks){
      writeLog(m_id + " updateCommitted:");
      AONTask task = (AONTask)tasks.getItem(m_taskID);
      boolean inRange = (getDistance(m_x,m_y,task.getX(),task.getY()) <
m_taskRange);
      task.setInRange(m_id,inRange);
      if(task.isExpired() || chooseToDropTask(task)){
         m_state = C_UNCOMMITTED;
         m_taskID=-1;
         m_plan.clear();
         if(m_skillHistoryLength <= 0){
            m_unfilledSkillsCount =
task.countUnfilledSkills(m_unfilledSkillsCount);
         }else{

m_unfilledSkillHistory.addSkillShortage(task.countUnfilledSkills());
         }//else
         task.removeAgentCommitted(m_id);
         writeLog(m_id + " task failed " + task);
      }else if(task.skillRequirementsFilled()){
         m_numberOfSuccessfulTeamsJoined++;
         m_state = C_ACTIVE;
         m_activeTimeRemaining = task.getDuration();
      }else{
         if(m_committedTimeRemaining > 0){
            m_committedTimeRemaining--;
         }//if
         if(m_committedTimeRemaining <= 0){
            if(task.agentsNeeded() == 1 || getLength(m_plan) > 0){
               m_committedTimeRemaining = m_maxCommittedTime;
            }else{
               task.removeAgentCommitted(m_id);
               m_state = C_UNCOMMITTED;
               m_taskID=-1;
            }//else
         }//if
      }//else
   }//updateCommitted

   /* This function updates the agent when its state is active
*/
   public void updateActive(){
```

```
      writeLog(m_id + " updateActive:");
      if(m_activeTimeRemaining > 0){
         m_activeTimeRemaining--;
      }//if
      if(m_activeTimeRemaining <= 0){
         m_state = C_UNCOMMITTED;
         m_taskID = -1;
      }else{
         m_lastWorkCatagory = C_WORKED;
      }//else
   }//updateActive

   /* This function increments the life span of the agent's neighbor
*/
   /*   connections
*/
   public void incrementNeighborConnections(){
      for(int i = 0; i < m_neighbors.getLength(); i++){
         int life = m_neighbors.getItem(i).connectionLife();
         m_neighbors.setConnectionLife(i,life+1);
      }//for
   }//incrementNeighborConnections

   /* This function updates the rating for each of the agent's neighbors
*/
   public void updateNeighborRatings(){
      int totalTime = 0;
      for(int i = 0; i < m_neighbors.getLength(); i++){
         int life = m_neighbors.getItem(i).connectionLife();
         totalTime += life;
      }//for
      for(int i = 0; i < m_neighbors.getLength(); i++){
         int life = m_neighbors.getItem(i).connectionLife();
         float rating = (float)life/totalTime;
         m_neighbors.setRating(i,rating);
      }//for
   }//updateNeighborRatings

   /* This function is used to find the best agent recommendation for a new
*/
   /*   neighbor by doing a recursive search of the network until depth has
*/
   /*   reached 0. The best agent recommendation is returned back to the
*/
   /*   agent who originated the request for agent recommendations
*/
   public AgentRecommendation recommendNeighbor(int depth, Vector
agentsVisited, AgentX sourceAgent){
      int bestAgentID = -1, bestAgentDegree = 0;
      depth--;
      agentsVisited.add(m_id);
      if(depth <= 0){
         if(!sourceAgent.hasNeighbor(m_id) && m_id != sourceAgent.getID()){
            bestAgentID = m_id;
            bestAgentDegree = m_neighbors.getLength();
         }//if
      }else{
         for(int i = 0; i < m_neighbors.getLength(); i++){
            int id = m_neighbors.getItem(i).agentID();
            if(!contains(id,agentsVisited)){
               AgentX neighbor = (AgentX)getAgent(id);
               AgentRecommendation agentRecommendation =
neighbor.recommendNeighbor(depth,agentsVisited,sourceAgent);
```

```
               if(agentRecommendation.getID() >= 0){
                  if((int)agentRecommendation.getValue() > bestAgentDegree &&
(int)agentRecommendation.getValue() < m_maxDegree){
                     bestAgentID = agentRecommendation.getID();
                     bestAgentDegree = (int)agentRecommendation.getValue();
                  }//if
               }//if
            }//if
         }//for
         if(!sourceAgent.hasNeighbor(m_id) && m_id != sourceAgent.getID()){
            if(bestAgentID == -1){
               bestAgentID = m_id;
               bestAgentDegree = m_neighbors.getLength();
            }else{
               if(m_neighbors.getLength() > bestAgentDegree){
                  bestAgentID = m_id;
                  bestAgentDegree = m_neighbors.getLength();
               }//if
            }//else
         }//if
      }//else
      return new AgentRecommendation(bestAgentID,bestAgentDegree);
   }//recommendNeighbor

   /* This function returns the agent id of the most highly recommended
*/
   /*   agent who may become this agent's new neighbor
*/
   public int pickNewNeighbor(){
      AgentRecommendation recommended =
recommendNeighbor(m_communicationDepth+1,new Vector(),this);
      return recommended.getID();
   }//pickNewNeighbor

   /* This function returns whether the agent has chosen to join a team
*/
   /*   by committing to a task that another agent has proposed
*/
   public boolean joinTeam(AONTaskList tasks){
      AONTask task = pickNeighborTask(tasks);
      if(task != null){
         m_lastWorkCatagory = C_JOINED;
         boolean isImpossible =
task.taskImpossible(m_neighborhoodStats.getSkillCount());
         if(isImpossible){
            m_joinedImpossibleTaskCount++;
         }//if
      }else{
         task = pickTask(tasks);
         if (task!=null){
            m_lastWorkCatagory = C_PROPOSED;
            boolean isImpossible =
task.taskImpossible(m_neighborhoodStats.getSkillCount());
            if(isImpossible){
               m_proposedImpossibleTaskCount++;
            }//if
         }//if
      }//else

      if(task != null){
         if(acceptTask(task)){
            writeLog(m_id + " " + m_lastWorkCatagory + " joins task " +
task.toString());
```

```
            m_committedTimeRemaining = m_maxCommittedTime;
            m_numberOfTeamsJoined++;

            if(m_teamsJoinedHistory.contains(task.getID())){
               m_repeatTaskCount++;
            }//if
            m_teamsJoinedHistory.add(task.getID());
            return true;
         }//if
      }//if
      return false;
   }//joinTeam

   /* This function returns true if the agent has accepted the specified
*/
   /*   task
*/
   public boolean acceptTask(AONTask task){
      if(task == null){
         writeLog(m_id  + "tried to accept null task");
         return false;
      }//if

      // Assuming each agent has a single skill
      int skill = task.findSkillIndex(m_skill);
      if(skill >= 0){
         task.addAgentCommitted(m_id,skill);
         m_color = task.getColor();
         m_taskID = task.m_id;
         m_agentTask = task;
         m_teamString = task.getTeamString();
         m_state = C_COMMITTED;
         if(m_useRoads){
            plan();
         }//if
         return true;
      }//if
      return false;
   }//acceptTask

   /* This function returns the percentage of the agent's neighbors who are
*/
   /*    not committed to a task
*/
   public float percentNeighborsUncommitted(){
      if(m_neighbors.getLength() == 0){
         return 0;
      }//if
      int sumUncommitted = 0;
      for(int i = 0; i < m_neighbors.getLength(); i++){
         int id = m_neighbors.getItem(i).agentID();
         //Abstract into function that allows choice between touching other
agent and sending message
         AgentX neighbor = getAgent(id);
         if(neighbor != null){
            if(neighbor.getState() == C_UNCOMMITTED){
               sumUncommitted++;
            }//if
         }//if
      }//for
      return (float)sumUncommitted/m_neighbors.getLength();
   }//percentNeighborsUncommitted
```

```
   /* This function returns the index of the first unfilled skill in the
*/
   /*   specified task, or -1 if all skills are filled in the task
*/
   public int skillNeededInTask(AONTask task){
      Vector skillsRequired = task.getSkillsRequired();
      for(int j = 0; j < getLength(skillsRequired); j++){
         int requiredSkill = ((Integer)getItem(j,skillsRequired)).intValue();
         if(m_skill == requiredSkill){
            Vector agentsCommitted = task.getAgentsCommitted();
            int committedAgentID =
((Integer)getItem(j,agentsCommitted)).intValue();
            if(committedAgentID < 0){
               return j;
            }//if
         }//if
      }//for
      return -1;
   }//skillNeededInTask

   /* This function returns the index of the first neighbor found to be
*/
   /*   committed to the specified task, or -1 if no neighbor is committed
*/
   /*   to the specified task
*/
   public int neighborCommitted(AONTask task){
      Vector agentsCommitted = task.getAgentsCommitted();
      for(int i = 0; i < m_neighbors.getLength(); i++){
         Integer neighborID = m_neighbors.getItem(i).agentID();
         if(contains(neighborID,agentsCommitted)){
            return i;
         }//if
      }//for
      return -1;
   }//neighborCommitted

   /* This function returns the first task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickNeighborTask(AONTaskList tasks){
      for(int i = 0; i < m_neighbors.getLength(); i++){
         Integer neighborID = m_neighbors.getItem(i).agentID();
         AgentX neighbor = getAgent(neighborID);
         if(neighbor.m_taskID >= 0){
            AONTask task = (AONTask)tasks.getItem(neighbor.m_taskID);
            if(!task.isExpired() && task.isNeeded(m_skill)){
               return task;
            }//if
         }//if
      }//for
      return null;
   }//pickNeighborTask

   /* This function returns the task with the best percentage of skills
*/
   /*   that are filled (the closest to being actively worked on)
*/
   public AONTask pickBestNeighborTask(AONTaskList tasks){
      AONTask bestTask = null;
      float bestRatio = 0.25f;
      for(int i = 0; i < m_neighbors.getLength(); i++){
```

```
         Integer neighborID = m_neighbors.getItem(i).agentID();
         AgentX neighbor = getAgent(neighborID);
         if(neighbor.m_taskID >= 0){
            AONTask task = (AONTask)tasks.getItem(neighbor.m_taskID);
            if(!task.isExpired() && task.isNeeded(m_skill)){
               float committedRatio = task.getAgentsCommittedRatio();
               float ratio = skillMatchRatio(task);
               if(ratio >= bestRatio){
                  bestTask = task;
                  bestRatio = ratio;
               }//if
            }//if
         }//if
      }//for
      return bestTask;
   }//pickBestNeighborTask

   /* This function updates the neighborhood statistics for the agent's
*/
   /*   neighborhood
*/
   private void updateNeighborhoodStats(){
      m_neighborhoodStatsUpdateCount++;
      if(m_neighborhoodStats == null || (m_neighborhoodStatsUpdateCount >= 10
&& m_rewired)){
         m_rewired = false;
         m_neighborhoodStatsUpdateCount = 0;
         m_neighborhoodStats = getNeighborhoodStats();

         int skillsFound = 0;
         m_neighborhoodSkillDistance = 0.0f;
         m_neighborhoodSkillShortage = 0;
         Vector<Integer> neighborhoodSkillCount =
m_neighborhoodStats.getSkillCount();
         Vector<Integer> neighborhoodSkillDistance =
m_neighborhoodStats.getSkillDistance();
         for(int i = 0; i < m_systemSkillCount; i++){
            int count = (Integer)getItem(i,neighborhoodSkillCount);
            if(count <= 0){
               m_neighborhoodSkillShortage++;
            }//if
            int distance = (Integer)getItem(i,neighborhoodSkillDistance);
            if(distance >= 0){
               skillsFound++;
               m_neighborhoodSkillDistance += distance;
            }//if
         }//for
         if(skillsFound > 0){
            m_neighborhoodSkillDistance =
m_neighborhoodSkillDistance/skillsFound;
         }//if
      }//if
   }//updateNeighborhoodStats

   /* This function initializes the unfilled skill count vector
*/
   private void initializeUnfilledSkillCount(){
      if(m_skillHistoryLength <= 0){
         if(m_unfilledSkillsCount == null){
            m_unfilledSkillsCount = new Vector();
            for(int i = 0; i < m_systemSkillCount; i++){
               m_unfilledSkillsCount.add(0);
            }//for
```

```
         }//if
      }else{
         if(m_unfilledSkillHistory == null){
            m_unfilledSkillHistory = new
SkillShortageHistory(m_systemSkillCount,m_skillHistoryLength);
         }//if
      }//else
   }//initializeUnfilledSkillCount

   /* This function returns an array containing the number of agents in the
*/
   /*   neighbor list that possess each skill
*/
   public int [] getNeighborTally(){
      int [] tally = new int [parent.getSkillMax()];
      for(int i = 0; i < tally.length; i++)tally[i] = 0;
      for(int i = 0; i < m_neighbors.getLength(); i++){
         int id = m_neighbors.getItem(i).agentID();
         AgentX neighbor = (AgentX)getAgent(id);
         if(neighbor != null){
            if(neighbor.isAvailable()){
               if(neighbor.m_skill < tally.length)tally[neighbor.m_skill]++;
            }//if
         }//if
      }//for

      tally[m_skill]++;
      return tally;
   }//getNeighborTally

   /* This function returns the first task which this agent is needed for
*/
   public AONTask pickTask(AONTaskList tasks){
      AONTask bestTask = null;
      int ctAvail = 0;
      int ctNeeded=0;
      for(int i = tasks.getFirstCurrentIndex(); i < tasks.getLength(); i++){
         AONTask task = (AONTask)tasks.getItem(i);
         boolean isAvail = !task.isTaken();
         boolean isNeeded = task.isNeeded(m_skill);
         if(isAvail && isNeeded){
            return task;
         }//if
      }//for
      return null;
   }//pickTask

   /* This function returns the best task for this agent based on the skills
*/
   /*   already filled in the task and the skill to be filled by the agent
*/
   public AONTask pickBestTask(AONTaskList tasks){
      AONTask bestTask = null;
      float bestRatio = .34f;
      int ctAvail = 0;
      int ctNeeded=0;
      for(int i = tasks.getFirstCurrentIndex(); i < tasks.getLength(); i++){
         AONTask task = (AONTask)tasks.getItem(i);
         boolean isAvail = !task.isTaken();
         boolean isNeeded = task.isNeeded(m_skill);
         if(isAvail)ctAvail++;
         if(isAvail && isNeeded){
            ctNeeded++;
```

```
            float ratio = skillMatchRatio(task);
            if(ratio >= bestRatio){
               bestTask = task;
               bestRatio = ratio;
            }//if
         }//if
      }//for
      return bestTask;
   }//pickBestTask

   /* This function returns the percentage of skills filled in the specified
*/
   /*   task by the skills available in the agents who are neighbors of this
*/
   /*   agent
*/
   public float skillMatchRatio(AONTask task){
      int[]need = task.getSkillTally();
      int[]avail = getNeighborTally();
      int totalNeed=0;
      float totalHave =0.0f;
      for(int i = 0; i < need.length;i++){
         totalNeed+=need[i];
         totalHave += Math.min(need[i], avail[i]);
      }//for

      return totalHave/totalNeed;
   }//skillMatchRatio

   /* This function updates the spring adjusted x and y pixel positions for
*/
   /*   this agent based on simulated spring forces on the connections to its
*/
   /*   neighbors
*/
   public void adjustLocation(int maxX, int maxY){
      if(m_moveToTask && m_state == C_COMMITTED){
         if(m_useRoads){
            moveTowardsTaskOnRoads(maxX,maxY);
         }else{
            moveTowardsTask(maxX,maxY);
         }//else
      }//if

      float delta_x = 0.0f, delta_y = 0.0f;
      for(int i = 0; i < m_neighbors.getLength(); i++){
         AgentX neighbor = getAgent(m_neighbors.getItem(i).agentID());
         if(neighbor != null){
            int lengthX = neighbor.getX()-m_springX;
            int lengthY = neighbor.getY()-m_springY;
            float ratioX = lengthX/(float)maxX;
            float ratioY = lengthY/(float)maxY;
            delta_x = delta_x + getSpringEdgeChange(ratioX, (float)maxX, 0.8f);
            delta_y = delta_y + getSpringEdgeChange(ratioY, (float)maxY, 0.8f);
         }//if
      }//for

      delta_x = validateDelta(delta_x);
      delta_y = validateDelta(delta_y);
      m_springX = m_springX + (int)delta_x;
      m_springY = m_springY + (int)delta_y;

      if(m_springX < 0)m_springX = 0;
```

```
      else if(m_springX > maxX)m_springX = maxX;

      if(m_springY < 0)m_springY = 0;
      else if(m_springY > maxY)m_springY = maxY;
   }//adjustLocation

   /* This function makse sure the delta x or y pixel change is within the
*/
   /*   the range of +-maxDelta pixels (50 in this case)                  */
   private float validateDelta(float delta){
      float maxDelta = 50.0f;
      if(delta > maxDelta){
         delta = maxDelta;
      }else if(delta < -1*maxDelta){
         delta = -1*maxDelta;
      }//else
      return delta;
   }//validateDelta

   /* This function returns the spring simulated change for either x or y
*/
   /*   pixel location for one edge to a neighboring agent
*/
   public float getSpringEdgeChange(float lengthRatio, float max, float
springConstant){
      float sign = 1.0f;
      if(lengthRatio < 0.0f){
         sign = -1.0f;
      }//if

      float value = sign*springConstant*lengthRatio*lengthRatio*max;
      return value;
   }//getSpringEdgeChange

   /* This function moves the agent location towards the task the agent is
*/
   /*   committed to by travelling in a straight line
*/
   private void moveTowardsTask(int maxX, int maxY){
      if(m_taskID >= 0){
         if(m_taskRange <
getDistance(m_x,m_y,m_agentTask.getX(),m_agentTask.getY())){
            int lengthX = m_agentTask.getX()-m_x;
            int lengthY = m_agentTask.getY()-m_y;
            if(lengthX > 50){
               lengthY = lengthY*(50/lengthX);
               lengthX = 50;
            }else if(lengthX < -50){
               lengthY = lengthY*Math.abs(50/lengthX);
               lengthX = -50;
            }//else if
            if(lengthY > 50){
               lengthX = lengthX*(50/lengthY);
               lengthY = 50;
            }else if(lengthY < -50){
               lengthX = lengthX*Math.abs(50/lengthY);
               lengthY = -50;
            }//else if
            m_x = m_x + lengthX;
            m_y = m_y + lengthY;
            if(m_x < 0)m_x = 0;
            else if(m_x > maxX)m_x = maxX;
```

```
            if(m_y < 0)m_y = 0;
            else if(m_y > maxY)m_y = maxY;
         }//if
      }//if
   }//moveTowardsTask

   /* This function moves the agent location towards the task the agent is
*/
   /*   committed to by moving it along the roads defined in m_roads object
*/
   public void moveTowardsTaskOnRoads(int maxX, int maxY){
      int maxDistance = 70;
      int distanceLeft = maxDistance;
      int distanceTraveled = 0;
      m_distanceTravelled = 0;
      boolean timeExpired = false;
      int planPointsUsed = 1;
      long startTime = System.currentTimeMillis();
      while(distanceTraveled < maxDistance && m_plan.size() > 0 &&
!timeExpired){
         Point nextPoint = (Point)m_plan.elementAt(0);
         int x = (int)nextPoint.getX(), y = (int)nextPoint.getY();
         int distance = getDistance(m_x,m_y,x,y);
         if(distance > 0){
            float lengthX = x-m_x;
            float lengthY = y-m_y;

            if(distance-distanceLeft > 0){
               lengthX = lengthX*((float)distanceLeft/distance);
               lengthY = lengthY*((float)distanceLeft/distance);
            }//if

            m_x = m_x + (int)lengthX;
            m_y = m_y + (int)lengthY;

            int currentDistance = (int)Math.sqrt((lengthX*lengthX) +
(lengthY*lengthY));
            distanceTraveled = distanceTraveled + currentDistance;
            distanceLeft = distanceLeft - currentDistance;
         }//if
         distance = getDistance(m_x,m_y,x,y);
         if(distance == 0){
            if(m_plan.size() > 1){
               m_shapeID =
m_roads.getShape(m_plan.elementAt(0),m_plan.elementAt(1));
            }//if
            m_plan.remove(0);
            planPointsUsed++;
         }//if
         if(System.currentTimeMillis()-startTime > m_maximumPlanTime){
            System.err.println("Possible Infinite Loop:
moveTowardsTaskOnRoads\n    Agent=" + m_id + " shape=" + m_shapeID + "
distanceTraveled=" + distanceTraveled + " distance=" + distance + "
planLength=" + m_plan.size() + " planPointsUsed=" + planPointsUsed);
            System.out.println("Possible Infinite Loop:
moveTowardsTaskOnRoads\n    Agent=" + m_id + " shape=" + m_shapeID + "
distanceTraveled=" + distanceTraveled + " distance=" + distance + "
planLength=" + m_plan.size() + " planPointsUsed=" + planPointsUsed);
            timeExpired = true;
         }//if
      }//while

      m_distanceTravelled = m_distanceTravelled + distanceTraveled;
```

```
   }//moveTowardsTaskOnRoads

   /* This function gets a plan for moving towards the task the agent is
*/
   /*   committed to by moving along the roads defined in the m_roads object
*/
   public void plan(){
      m_plan =
m_roads.getPlan(m_x,m_y,m_shapeID,m_agentTask.getX(),m_agentTask.getY(),m_taskR
ange,100);
   }//plan

   /* This function returns the number of neighbors this agent has who are
*/
   /*   not committed to a task
*/
   public int getUnCommittedNeighborCount(){
      int count = 0;
      for(int i = 0; i < m_neighbors.getLength(); i++){
         int id = m_neighbors.getItem(i).agentID();
         AgentX neighbor = getAgent(id);
         if(neighbor != null){
            if(neighbor.getState() == C_UNCOMMITTED){
               count++;
            }//if
         }//if
      }//for
      return count;
   }//getUnCommittedNeighborCount

   /* This function returns the number of neighbors this agent has who are
*/
   /*   committed to a task
*/
   public int getCommittedNeighborCount(){
      int count = 0;
      for(int i = 0; i < m_neighbors.getLength(); i++){
         int id = m_neighbors.getItem(i).agentID();
         AgentX neighbor = getAgent(id);
         if(neighbor != null){
            if(neighbor.getState() == C_COMMITTED){
               count++;
            }//if
         }//if
      }//for
      return count;
   }//getCommittedNeighborCount

   /* This function returns the number of neighbors this agent has who are
*/
   /*   actively working on the task they are committed to
*/
   public int getActiveNeighborCount(){
      int count = 0;
      for(int i = 0; i < m_neighbors.getLength(); i++){
         int id = m_neighbors.getItem(i).agentID();
         AgentX neighbor = getAgent(id);
         if(neighbor != null){
            if(neighbor.getState() == C_ACTIVE){
               count++;
            }//if
         }//if
      }//for
```

```
      return count;
   }//getActiveNeighborCount

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      return false;
   }//chooseToDropTask

   /* This function returns a skill comparison weight between 0.0 and 1.0
*/
   /*   A value of 0.0 means their is no overlap in the skill sets
*/
   /*   A value of 1.0 means the skill sets are disjoint
*/
   public float getSkillComparison(int skill){
      if(skill == m_skill)return 0.0f;
      return 1.0f;
   }//getSkillComparison

   /* This function returns whether or not the agent has decided to adapt its
*/
   /*   network connections to its neighbors
*/
   public boolean chooseToAdapt(){
      boolean adapt = false;
      float num = (float)m_adaptationRate/getAgentCount();
      if (rand== null){
         writeLog(m_id + "Bad rand");
      }//if
      float random = rand.nextFloat();
      if(random < num && m_adaptNetwork){
         adapt = true;
      }//if
      return adapt;
   }//chooseToAdapt

   /* This function changes the agents network connections by getting a
*/
   /*   recommendation of a new neighbor from its current neighbors, then
*/
   /*   removing a connection to one of its current neighbors
*/
   public int rewire(){
      if(m_neighbors.getLength()==0) return -1;
      int newNeighbor = pickNewNeighbor();
      if(newNeighbor >= 0){
         if(!addNeighbor(newNeighbor)){
            newNeighbor = -1;
         }//if
      }//if
      if(newNeighbor >= 0){
         int oldNeighborID = pickRemoveNeighbor();
         removeNeighbor(oldNeighborID);
         resetTaskJoinCount();
         m_edgesChanged++;
         m_lastWorkCatagory = C_REWIRED;
         m_rewired = true;
      }//if
      return newNeighbor;
   }//rewire
```

```
   /* This function resets the statistics kept about tasks neighbors have
*/
   /*   joined (Implemented by Performance agent)
*/
   public void resetTaskJoinCount(){

   }//resetTaskJoinCount

   /* This function draws the agent as a filled oval using the specified
*/
   /*   graphics object at the location determined by the x and y offsets
*/
   /*   and the agent's location and drawing the task id the agent is
*/
   /*   committed to when specified and drawing the agent id when specified
*/
   public void paintAgent(int columnOffset, int rowOffset, int ovalWidth,
boolean drawTaskIDs, boolean drawAgentIDs, Graphics g){
      Point agentLocation = new Point(getX(),getY());
      int agentX = (int)agentLocation.getX(), agentY =
(int)agentLocation.getY();
      int numNeighbors = m_neighbors.getLength();
      for(int j = 0; j < numNeighbors; j++){
         int id = m_neighbors.getItem(j).agentID();
         AgentX neighbor = getAgent(id);
         g.setColor(getNeighborColor(j));
         Point neighborLocation = new Point(neighbor.getX(),neighbor.getY());
         int x1 = agentX+columnOffset, y1 = agentY+rowOffset;
         int x2 = (int)neighbor.getX()+columnOffset, y2 =
(int)neighbor.getY()+rowOffset;
         g.drawLine(x1,y1,x2,y2);
      }//for

      g.setColor(getColor());
      int ovalXOffset = columnOffset-((int)ovalWidth/2), ovalYOffset =
rowOffset-((int)ovalWidth/2);
      g.fillOval(agentX+ovalXOffset,agentY+ovalYOffset,ovalWidth,ovalWidth);
      int state = getState();
      if((state == C_ACTIVE || state == C_COMMITTED) && drawTaskIDs){
         g.setColor(Color.black);
         int textXOffset = columnOffset+ovalWidth/2, textYOffset =
rowOffset+ovalWidth/2;
         g.drawString(getTeamString(),agentX+textXOffset,agentY+textYOffset);
      }//if
      if(drawAgentIDs){
         int textXOffset2 = columnOffset-ovalWidth/2, textYOffset2 = rowOffset-
ovalWidth/2;
         g.setColor(Color.green);

g.drawString(String.valueOf(getID()),agentX+textXOffset2,agentY+textYOffset2);
      }//if
   }//paintAgent

   /* This function draws the agent as a filled oval using the specified
*/
   /*   graphics object at the location determined by the x and y offsets
*/
   /*   and the agent's spring adjusted location and drawing the task id
*/
   /*   the agent is committed to when specified and drawing the agent id
*/
```

```
   /*   when specified
*/
   public void paintAgentSpringLayout(int columnOffset, int rowOffset, int
ovalWidth, boolean drawTaskIDs, boolean drawAgentIDs, Graphics g){
      Point agentLocation = new Point(getSpringX(),getSpringY());
      int agentX = (int)agentLocation.getX(), agentY =
(int)agentLocation.getY();
      int numNeighbors = m_neighbors.getLength();
      for(int j = 0; j < numNeighbors; j++){
         int id = m_neighbors.getItem(j).agentID();
         AgentX neighbor = getAgent(id);
         g.setColor(getNeighborColor(j));
         Point neighborLocation = new
Point(neighbor.getSpringX(),neighbor.getSpringY());
         int x1 = agentX+columnOffset, y1 = agentY+rowOffset;
         int x2 = (int)neighbor.getSpringX()+columnOffset, y2 =
(int)neighbor.getSpringY()+rowOffset;
         g.drawLine(x1,y1,x2,y2);
      }//for

      g.setColor(getColor());
      int ovalXOffset = columnOffset-((int)ovalWidth/2), ovalYOffset =
rowOffset-((int)ovalWidth/2);
      g.fillOval(agentX+ovalXOffset,agentY+ovalYOffset,ovalWidth,ovalWidth);
      int state = getState();
      if((state == C_ACTIVE || state == C_COMMITTED) && drawTaskIDs){
         g.setColor(Color.black);
         int textXOffset = columnOffset+ovalWidth/2, textYOffset =
rowOffset+ovalWidth/2;
         g.drawString(getTeamString(),agentX+textXOffset,agentY+textYOffset);
      }//if
      if(drawAgentIDs){
         int textXOffset2 = columnOffset-ovalWidth/2, textYOffset2 = rowOffset-
ovalWidth/2;
         g.setColor(Color.green);

g.drawString(String.valueOf(getID()),agentX+textXOffset2,agentY+textYOffset2);
      }//if
   }//paintAgentSpringLayout

   /* This function returns the agent id of the neighbor occupying the
*/
   /*   position of the specified index in the neighbor list
*/
   public int getNeighborID(int index){
      return m_neighbors.getItem(index).agentID();
   }//getNeighborID

   /* This function counts the number of skills present in the agent's
*/
   /*   neighborhood and the number of network links to each skill type
*/
   public NeighborhoodStats countNeighborSkills(int depth, Vector
agentsVisited, NeighborhoodStats stats, AgentX sourceAgent){
      depth--;
      agentsVisited.add(m_id);
      if(depth > 0){
         for(int i = 0; i < m_neighbors.getLength(); i++){
            int id = m_neighbors.getItem(i).agentID();
            if(!contains(id,agentsVisited)){
               AgentX neighbor = (AgentX)getAgent(id);
               stats =
neighbor.countNeighborSkills(depth,agentsVisited,stats,sourceAgent);
```

```
            }//if
         }//for
      }//if
      stats.addNeighborStats(m_skill,m_neighborhoodRadius-
depth,getPerformance());
      return stats;
   }//countNeighborSkills

   /* This function gets the neighborhood statistics for the agent's
*/
   /*   neighborhood
*/
   public NeighborhoodStats getNeighborhoodStats(){
      NeighborhoodStats stats = new NeighborhoodStats(m_systemSkillCount);
      return countNeighborSkills(m_neighborhoodRadius,new Vector(),stats,this);
   }//getNeighborhoodStats

   /* This function resets the agent's neighborhood statistics to null
*/
   /*   which will cause the agent to create a fresh set of neighborhood
*/
   /*   statistics the next time the agent is updated
*/
   public void resetNeighborhoodStats(){
      m_neighborhoodStats = null;
   }//resetNeighborhoodStats

   /* This function sets the number of teams the agent has joined
*/
   /*   (For debug and performance validation purposes)
*/
   public void setTeamsJoined(int teamsJoined){
      m_numberOfTeamsJoined = teamsJoined;
   }//setTeamsJoined

   /* This function sets the number of successful teams the agent has joined
*/
   /*   (For debug and performance validation purposes)
*/
   public void setSuccessfulTeamsFormed(int successfulTeamsFormed){
      m_numberOfSuccessfulTeamsJoined = successfulTeamsFormed;
   }//setSuccesfulTeamsFormed

   /* This function sets the array of counters for how many time each skill
*/
   /*   has been unfilled in a failed task (For debug purposes)
*/
   public void setFailedSkillCount(Vector<Integer> failedSkillsCount){
      m_unfilledSkillsCount = failedSkillsCount;
   }//setFailedSkillCount

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String output = "";
      output = output + "Agent " + m_id + " [skill=" + m_skill + "]"
                  + "\n neighbors=" + m_neighbors
                  + "\n degree=" + m_neighbors.getLength()
                  + "\n neighborhoodSkillDistance=" +
m_neighborhoodSkillDistance;
      return output;
   }//toString
}//class AgentX
```

```
---------------------------------AgentX.java-------------------------------------
-

-------------------------AONMouseInputListener.java----------------------------
-

import java.util.*;
import java.awt.event.*;
import javax.swing.event.*;

/**
 * AONMouseInputListener Class
 *
 *
 */
public class AONMouseInputListener implements MouseInputListener{
   int m_x;                  /*
*/
   int m_y;                  /*
*/
   int m_xOffset;            /*
*/
   int m_yOffset;            /*
*/
   int m_maxError;           /*
*/
   Vector m_agents;          /*
*/
   AONTaskList m_tasks;      /*
*/
   TestAON m_testAON;        /*
*/

   /* constructor
*/
   public AONMouseInputListener(){
      m_agents = new Vector();
      m_tasks = new AONTaskList();
   }//constructor

   /* constructor
*/
   public AONMouseInputListener(int xOffset, int yOffset, Vector agents,
AONTaskList tasks, TestAON testAON){
      m_xOffset = xOffset;
      m_yOffset = yOffset;
      m_maxError = 10;
      m_agents = agents;
      m_tasks = tasks;
      m_testAON = testAON;
   }//constructor

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
```

```
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   public void mouseClicked(MouseEvent e){
   }//mouseClicked

   public void mouseEntered(MouseEvent e){
   }//mouseEntered

   public void mouseExited(MouseEvent e){
   }//mouseExited

   /* This function handles the mouse being pressed
*/
   public void mousePressed(MouseEvent e){
      if(m_testAON != null){
         int clickX = e.getX()-m_xOffset, clickY = e.getY()-m_yOffset;
         int minAgentIndex = -1, minDistance = 99999;
         for(int i = 0; i < getLength(m_agents); i++){
            AgentX agent = (AgentX)getItem(i,m_agents);
            int x, y;
            if(m_testAON.getDisplaySpringLayout()){
               x = agent.getSpringX();
               y = agent.getSpringY();
            }else{
               x = agent.getX();
               y = agent.getY();
            }//else
            int distance = getDistance(clickX,clickY,x,y);
            if(distance < m_maxError && distance < minDistance){
               minAgentIndex = i;
               minDistance = distance;
            }//if
         }//for
         int minTaskIndex = -1;
         minDistance = 99999;
         for(int i = m_tasks.getFirstCurrentIndex(); i < m_tasks.getLength();
i++){
            AONTask task = (AONTask)m_tasks.getItem(i);
            int x, y;
            x = task.getX();
            y = task.getY();
```

```
            int distance = getDistance(clickX,clickY,x,y);
            if(distance < m_maxError && distance < minDistance){
               minTaskIndex = i;
               minDistance = distance;
            }//if
         }//for

         if(minAgentIndex >= 0){
            AgentX closeAgent = (AgentX)getItem(minAgentIndex,m_agents);
            m_testAON.setSelectedAgent(closeAgent);
         }else{
            m_testAON.setSelectedAgent(null);
         }//else
         if(minTaskIndex >= 0){
            AONTask closeTask = (AONTask)m_tasks.getItem(minTaskIndex);
            m_testAON.setSelectedTask(closeTask);
         }else{
            m_testAON.setSelectedTask(null);
         }//else
         m_testAON.updateMessageFrame();
      }//if
   }//mousePressed

   public void mouseReleased(MouseEvent e){
   }//mouseReleased

   public void mouseDragged(MouseEvent e){
   }//mouseDragged

   public void mouseMoved(MouseEvent e){
   }//mouseMoved

   /* This function returns the distance between two points specified by
*/
   /*   (x1,y1) and (x2,y2)
*/
   private int getDistance(int x1, int y1, int x2, int y2){
      return (int)Math.sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
   }//getDistance
}//AONMouseInputListener

-------------------------AONMouseInputListener.java----------------------------
-

---------------------------AONReportCatagory.java-----------------------------
-

 import java.util.*;
import java.io.IOException;
import java.io.BufferedWriter;
import java.io.FileWriter;

/**
 * AONReportCatagory Class
 *
 *
 */
public class AONReportCatagory{
   String m_name;            /*
*/
   String m_filename;        /*
*/
```

```
   Vector m_lines;           /*
*/
   Vector m_currentLine;     /*
*/
   String m_delimeterString; /*
*/

   /* constructor
*/
   public AONReportCatagory(String name, String filename){
      m_name = name;
      m_filename = filename;
      m_lines = new Vector();
      m_currentLine = new Vector();
      m_delimeterString = ",";
   }//constructor

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   /* This function adds a value to the current line in the report
*/
   public void addValue(int index, String value){
      if(getLength(m_lines) > 0 && index < getLength(m_lines)){
         m_currentLine = (Vector)getItem(index,m_lines);
         m_currentLine.add(value);
         setItem(index,m_lines,m_currentLine);
      }else{
         m_currentLine.add(value);
      }//else
   }//addValue
```

```
   public String printLines(){
      return m_lines.toString();
   }//printLines

   public void newLine(){
      m_lines.add(m_currentLine);
      m_currentLine = new Vector();
   }//newLine

   public void replaceLine(int index){
      setItem(index,m_lines,m_currentLine);
      m_currentLine = new Vector();
   }//newLine

   public void save(){
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(m_filename));
//             out.write(toString());
         for(int i = 0; i < getLength(m_lines); i++){
            Vector lineValues = (Vector)getItem(i,m_lines);
            for(int j = 0; j < getLength(lineValues); j++){
               if(j > 0){
                  out.write(",");
               }//else
               out.write((String)getItem(j,lineValues));
            }//for
            out.newLine();
         }//for
         out.close();
         System.out.println("Wrote file: " + m_filename);
      }catch(IOException ex){
         System.out.println("Error writing file" + m_filename + ": " +
ex.getMessage());
      }//catch
   }//save

   public void save(String filename){
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(filename));
//             out.write(toString());
         for(int i = 0; i < getLength(m_lines); i++){
            Vector lineValues = (Vector)getItem(i,m_lines);
            for(int j = 0; j < getLength(lineValues); j++){
               if(j > 0){
                  out.write(",");
               }//else
               out.write((String)getItem(j,lineValues));
            }//for
            out.newLine();
         }//for
         out.close();
         System.out.println("Wrote file: " + filename);
      }catch(IOException ex){
         System.out.println("Error writing file" + filename + ": " +
ex.getMessage());
      }//catch
   }//save

   public void computeAverage(){
      for(int i = 0; i < getLength(m_lines); i++){
         Vector lineValues = (Vector)getItem(i,m_lines);
         if(i == 1){
            lineValues.add("average");
```

```
         }else if(i > 1){
            float sum = 0.0f;
            for(int j = 1; j < getLength(lineValues); j++){
               try{
                  float value =
Float.parseFloat((String)getItem(j,lineValues));
                  sum = sum + value;
               }catch(Exception ex){
                  System.out.println("Error getting value in computeAverage: "
+ ex.toString());
               }//catch
            }//for
            float average = (float)sum / (getLength(lineValues)-1);
            lineValues.add(String.valueOf(average));
         }//else if
         setItem(i,m_lines,lineValues);
         //System.out.println("Line " + i + ": " + lineValues);
      }//for
   }//computeAverage

   public void computeConfidence(){
      for(int i = 0; i < getLength(m_lines); i++){
         Vector lineValues = (Vector)getItem(i,m_lines);
         if(i == 1){
            lineValues.add("2*StdErr");
         }else if(i > 1){
            float sum = 0.0f;
            for(int j = 1; j < getLength(lineValues); j++){
               try{
                  float value =
Float.parseFloat((String)getItem(j,lineValues));
                  sum = sum + value;
               }catch(Exception ex){
                  System.out.println("Error getting value in computeAverage: "
+ ex.toString());
               }//catch
            }//for

            float average = (float)sum / (getLength(lineValues)-1);
            sum = 0.0f;
            for(int j = 1; j < getLength(lineValues); j++){
               try{
                  float value =
Float.parseFloat((String)getItem(j,lineValues));
                  sum = sum + (value-average)*(value-average);
               }catch(Exception ex){
                  System.out.println("Error getting value in computeAverage: "
+ ex.toString());
               }//catch
            }//for
            double dev = Math.sqrt(sum/(getLength(lineValues)-2));
            double err = 2*dev/Math.sqrt(getLength(lineValues)-1);
            //System.out.println(" Average"+ average + " deviation=" + dev +  "
error=" +err);
            lineValues.add(String.valueOf(err));
         }//else if
         setItem(i,m_lines,lineValues);
         //System.out.println("Line " + i + ": " + lineValues);
      }//for
   }//computeConfidence

   public void computeImprovement(){
      int length = getLength(m_lines);
```

```
      int index = (int)(length / 2)-1;
      float stableValue = 0.0f;
      if(index >= 0){
         Vector line = (Vector)getItem(index,m_lines);
         stableValue = Float.parseFloat((String)getItem(getLength(line)-
2,line));
      }//if
      for(int i = 0; i < getLength(m_lines); i++){
         Vector lineValues = (Vector)getItem(i,m_lines);
         if(i == 1){
            lineValues.add("improvement");
         }else if(i > 1){
            try{
               float average =
Float.parseFloat((String)getItem(getLength(lineValues)-2,lineValues));
               float improvement = average - stableValue;
               lineValues.add(String.valueOf(improvement));
            }catch(Exception ex){
               System.out.println("Error getting value in computeImprovement: "
+ ex.toString());
            }//catch
         }//else if
         setItem(i,m_lines,lineValues);
      }//for
   }//computeImprovement

   public void computeImprovement(int minAdaptTimeStep){
      int length = getLength(m_lines);
      int index = getMinAdaptIndex(minAdaptTimeStep);
      float stableValue = 0.0f;
      if(index >= 0){
         Vector line = (Vector)getItem(index,m_lines);
         stableValue = Float.parseFloat((String)getItem(getLength(line)-
2,line));
      }//if
      for(int i = 0; i < getLength(m_lines); i++){
         Vector lineValues = (Vector)getItem(i,m_lines);
         if(i == 1){
            lineValues.add("improvement");
         }else if(i > 1){
            try{
               float average =
Float.parseFloat((String)getItem(getLength(lineValues)-2,lineValues));
               float improvement = average - stableValue;
               lineValues.add(String.valueOf(improvement));
            }catch(Exception ex){
               System.out.println("Error getting value in computeImprovement: "
+ ex.toString());
            }//catch
         }//else if
         setItem(i,m_lines,lineValues);
      }//for
   }//computeImprovement

   private int getMinAdaptIndex(int minAdaptTimeStep){
      int index = -1;
      for(int i = 0; i < getLength(m_lines) && index < 0; i++){
         Vector line = (Vector)getItem(i,m_lines);
         try{
            int timeStep = Integer.parseInt((String)getItem(0,line));
            if(timeStep == minAdaptTimeStep){
               index = i;
            }//if
```

```
         }catch(Exception ex){
         }//catch
      }//for
      return index;
   }//getMinAdaptIndex

   public String toString(){
      String value = "";
      for(int i = 0; i < getLength(m_lines); i++){
         m_currentLine = (Vector)getItem(i,m_lines);
         for(int j = 0; j < getLength(m_currentLine); j++){
            if(j > 0){
               value = value + m_delimeterString;
            }//if
            value = value + (String)getItem(j,m_currentLine);
         }//for
         value = value + "\n";
      }//for
      return value;
   }//toString

   public void pauseTest(String message){
      try{
         char input = ' ';
         byte temp[] = new byte[1];
         System.out.print("\n" + message + "\nPress <enter> to continue...");
         while(input != '\n'){
            System.in.read(temp);
            input = (char)temp[0];
         }//while
      }catch(IOException ex){
      }//catch
   }//pauseTest
}//AONReportCatagory
```

----------AONReportCatagory.java----------
-

----------AONReportModel.java----------
-

```
import java.util.*;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.FileReader;

/**
 * AONReportModel Class
 *
 *
 */
public class AONReportModel{
```

```
   boolean m_insertFirstColumn;
                              /*
*/
   Vector m_catagories;       /*
*/
   Vector m_catagoryNames;    /*
*/
   Vector m_catagoryFilenames;
                              /*
*/
   Vector m_catagoryTempValues;
                              /*
*/

   /* constructor
*/
   public AONReportModel(){
      m_insertFirstColumn = true;
      m_catagories = new Vector();
      m_catagoryNames = new Vector();
      m_catagoryFilenames = new Vector();
      m_catagoryTempValues = new Vector();
   }//constructor

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   public void addCategory(String columnName, String filename){
      m_catagories.add(new AONReportCatagory(columnName,filename));
      m_catagoryNames.add(columnName);
      m_catagoryFilenames.add(filename);
   }//addCategory
```

```
   public void computeAveragePerRun(){
      for(int i = 0; i < getLength(m_catagories); i++){
         AONReportCatagory c = (AONReportCatagory)getItem(i,m_catagories);
         c.computeAverage();
      }//for
   }//computeAveragePerRun

   public void computeImprovement(){
      for(int i = 0; i < getLength(m_catagories); i++){
         AONReportCatagory c = (AONReportCatagory)getItem(i,m_catagories);
         c.computeImprovement();
      }//for
   }//computeImprovement

   public void computeImprovement(int minAdaptTimeStep){
      for(int i = 0; i < getLength(m_catagories); i++){
         AONReportCatagory c = (AONReportCatagory)getItem(i,m_catagories);
         c.computeImprovement(minAdaptTimeStep);
      }//for
   }//computeImprovement

   public void computeConfidence(){
      for(int i = 0; i < getLength(m_catagories); i++){
         AONReportCatagory c = (AONReportCatagory)getItem(i,m_catagories);
         c.computeConfidence();
      }//for
   }//computeConfidence

   public void readFile(String filename){
      try{
         int index = 0;
         BufferedReader in = new BufferedReader(new FileReader(filename));
         String line = in.readLine();
         while(line != null){
            //System.out.println("Input line: " + line);
            parseLine(index,line);
            for(int i = 0; i < getLength(m_catagories); i++){
               AONReportCatagory c =
(AONReportCatagory)getItem(i,m_catagories);
               if(m_insertFirstColumn){
                  c.newLine();
               }else{
                  c.replaceLine(index);
               }//else
               //System.out.println("AONReportCatagory" + i + "\n" +
c.printLines());
               //c.save("DebugLine" + i + ".csv");
            }//for
            line = in.readLine();
            //System.out.println("performanceVector:
"+m_performanceLines.toString());
            //System.out.println("efficiencyVector:
"+m_efficiencyLines.toString());
            index++;
            //pauseTest("Line " + index + " in file " + filename + "...");
         }//while
         m_insertFirstColumn = false;
         //System.out.println("Closing file...");
         in.close();
      }catch(IOException ex){
         System.out.println("Error reading file " + filename + ": " +
ex.getMessage());
```

```
      }//catch
   }//readFile

   private void parseLine(int lineIndex, String line){
      int index = 0;
      String item = "";
      StringTokenizer splitString = new StringTokenizer(line,",");
      while(splitString.hasMoreTokens()){
         item = splitString.nextToken();
         //System.out.print(item + " ");
         if(index == 0 && m_insertFirstColumn || lineIndex == 0){
            for(int i = 0; i < getLength(m_catagories); i++){
               AONReportCatagory c =
(AONReportCatagory)getItem(i,m_catagories);
               c.addValue(lineIndex,item);
            }//for
         }else if(index > 0){
            AONReportCatagory c = (AONReportCatagory)getItem(index-
1,m_catagories);
            c.addValue(lineIndex,item);
         }//else
         index++;
      }//while
      //AONReportCatagory c = (AONReportCatagory)getItem(0,m_catagories);
      //System.out.println("AONReportCatagory 1: " + c.toString());
      //System.out.println("\nPerformance: " +
m_performanceLineValues.toString());
      //System.out.println("\nEfficiency: " +
m_efficiencyLineValues.toString());
   }//parseLine

   public void save(){
      for(int i = 0; i < getLength(m_catagories); i++){
         AONReportCatagory c = (AONReportCatagory)getItem(i,m_catagories);
         c.save();
      }//for
   }//save

   public void pauseTest(String message){
      try{
         char input = ' ';
         byte temp[] = new byte[1];
         System.out.print("\n" + message + "\nPress <enter> to continue...");
         while(input != '\n'){
            System.in.read(temp);
            input = (char)temp[0];
         }//while
      }catch(IOException ex){
      }//catch
   }//pauseTest
}//class AONReportModel

----------------------------AONReportModel.java---------------------------------
-

--------------------------------AONSkill.java-----------------------------------
-

import java.util.*;

/**
 * AONSkill Class
 *
```

```
 *
 */
public class AONSkill{
   int m_id;                 /*
*/
   int m_committedAgentID;   /*
*/

   int m_skillCount;    //counts how many of each skill are needed
                             /*
*/
   int m_skillsRemaining; //how many of each skill remain to be filled
                             /*
*/
   int m_totalSkillsRemaining; //total number of skills remaining
                             /*
*/
   int m_numberOfSkills;    //total possible skills [needed to set up
m_skillCt]
                             /*
*/

   /* constructor
*/
   public AONSkill(){
   }//constructor
}//AONSkill

--------------------------------AONSkill.java-----------------------------------
-

--------------------------------AONTask.java------------------------------------
-

import java.awt.*;
import java.util.*;

/**
 * AONTask Class
 *
 *
 */
class AONTask{
   final int C_FIREFIGHTER = 0, C_POLICE = 1, C_MEDIC = 2, C_UTILITY = 3,
C_LABOR = 4;
   final int C_PENDING = 0, C_SKILLS_FAIL = 1, C_DISTANCE_FAIL = 2,
C_TASK_COMPLETE = 3;
   TestAON parent;           /*
*/
   int m_id;                 /*
*/
   int m_x;                  /*
*/
   int m_y;                  /*
*/
   int m_state;              /*
*/
   int m_clusterIndex;       /*
*/
   int m_maxAdvertisedTime;  /*
*/
   int m_duration;           /*
*/
```

```
   int m_skillMax;           /*
*/
   int m_agentsCommittedCt;  /*
*/
   Vector<Float> m_skillPayoffs;
   Vector<Integer> m_skillsRequired;
   Vector<Integer> m_agentsCommitted;
   Vector<Boolean> m_agentsInRange;
   Vector<Float> m_skillWeights;
   Random rand;              /*
*/

   Color m_color;            /*
*/
   String m_teamString;      /* A unique identifying string for the team
*/

   boolean m_useCommandSkills;
                             /*
*/
   boolean m_useTaskRange;   /*
*/

   /* constructor
*/
   public AONTask(AONTask task){
      m_id = task.getID();
      m_x = task.getX();
      m_y = task.getY();
      m_clusterIndex = -1;

      m_maxAdvertisedTime = task.getMaxAdvertisedTime();
      m_duration = task.getDuration();

      m_skillPayoffs = new Vector();
      m_skillsRequired = (Vector)task.getSkillsRequired().clone();
      m_agentsCommitted = (Vector)task.getAgentsCommitted().clone();
      m_agentsInRange = new Vector();
      m_agentsCommittedCt=0;
      m_skillMax = task.getSkillMax();

      rand = new Random (System.currentTimeMillis()*m_id);
      m_color = new Color(1.0f,0.0f,0.0f);
      m_teamString = "0";
   }//constructor

   /* constructor
*/
   public AONTask(TestAON p, int id, int x, int y, int maxAdvertisedTime, int
duration, int skillMax){
      m_id = id;
      m_x = x;
      m_y = y;
      m_clusterIndex = -1;
      parent = p;
      m_maxAdvertisedTime = maxAdvertisedTime;
      m_duration = duration;
      m_skillPayoffs = new Vector();
      m_skillsRequired = new Vector();
      m_agentsCommitted = new Vector();
      m_agentsInRange = new Vector();
      m_agentsCommittedCt=0;
      m_skillMax = skillMax;
```

```
      rand = new Random (System.currentTimeMillis()*m_id);
      m_color = new Color(rand.nextFloat(),rand.nextFloat(),rand.nextFloat());
      m_teamString = String.valueOf(id);

      m_useCommandSkills = false;
      m_useTaskRange = false;
   }//constructor

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   public int getSkillMax(){
      return m_skillMax;
   }//getSkillMax

   public void setUseCommandSkills(boolean useCommandSkills){
      m_useCommandSkills = useCommandSkills;
   }//setUseCommandSkills

   public void setUseTaskRange(boolean useTaskRange){
      m_useTaskRange = useTaskRange;
   }//setUseTaskRange

   public void setRandomSeed(int randomSeed){
      rand = new Random(randomSeed);
   }//setRandomSeed

   public void addSkill(int skill, float payoff){
      m_skillPayoffs.add(payoff);
      m_skillsRequired.add(skill);
      m_agentsCommitted.add(-1);
```

```
      m_agentsInRange.add(!m_useTaskRange);
   }//addSkill

   public void addRandomSkills(int skillsPerTask, int skillCount){
      //Get random skills
      for(int i = 0; i < skillsPerTask; i++){
         m_skillPayoffs.add((float)rand.nextInt(10));
         m_skillsRequired.add(rand.nextInt(skillCount));
         m_agentsCommitted.add(-1);
         m_agentsInRange.add(!m_useTaskRange);
      }//for
   }//addRandomSkills

   public void addSkills(int skillsPerTask, int skillCount){
      int startSkillIndex = 0;

      if(m_useCommandSkills){
         //Add command skill first
         m_skillPayoffs.add((float)rand.nextInt(10));
         m_skillsRequired.add(0);
         m_agentsCommitted.add(-1);
         m_agentsInRange.add(true);//The Command Agent Doesn't have to move to
the task, so it is initially in range
         startSkillIndex = 1;
      }//if

      //Add other skills
      for(int i = startSkillIndex; i < skillsPerTask; i++){
         m_skillPayoffs.add((float)rand.nextInt(10));
         if(m_skillWeights == null){
            int skill;       /*
*/
            if(m_useCommandSkills){
               skill = rand.nextInt(skillCount-1)+1;//exclude command skill
            }else{
               skill = rand.nextInt(skillCount);
            }//if
            m_skillsRequired.add(skill);
         }else{
            m_skillsRequired.add(getWeightedSkill());
         }//else
         m_agentsCommitted.add(-1);
         m_agentsInRange.add(!m_useTaskRange);
      }//for
   }//addSkills

   public Vector<Integer> countSkills(Vector<Integer> skillCount){
      for(int i = 0; i < getLength(m_skillsRequired); i++){
         int skillID = (Integer)getItem(i,m_skillsRequired);
         int count = (Integer)getItem(skillID,skillCount);
         skillCount.setElementAt(count+1,skillID);
      }//for
      return skillCount;
   }//countSkills

   public int getWeightedSkill(){
      int fail_count = 0;
      Vector<Integer> skillWeightRandomOrder = new Vector();
      Vector<Integer> nonZeroWeights = new Vector();
      for(int i = 0; i < getLength(m_skillWeights); i++){
         float weight = ((Float)getItem(i,m_skillWeights)).floatValue();
         if(weight > 0.0){
            nonZeroWeights.add(i);
```

```
         }//if
      }//for
      for(int i = 0; i < getLength(nonZeroWeights); i++){
         int index = rand.nextInt(getLength(nonZeroWeights));
         index = ((Integer)getItem(index,nonZeroWeights)).intValue();
         while(skillWeightRandomOrder.contains(index)){
            index = rand.nextInt(getLength(nonZeroWeights));
            index = ((Integer)getItem(index,nonZeroWeights)).intValue();
         }//while
         skillWeightRandomOrder.add(index);
      }//for

      while(fail_count < 100){
         for(int i = 0; i < getLength(skillWeightRandomOrder); i++){
            float value = rand.nextFloat();
            int index =
((Integer)getItem(i,skillWeightRandomOrder)).intValue();
            float weight = ((Float)getItem(index,m_skillWeights)).floatValue();
            if(value < weight){
               return index;
            }//if
         }//for
         fail_count++;
      }//while
      return getLength(m_skillWeights)-1;
   }//getWeightedSkill

   public void setSkillWeights(Vector<Float> skillWeights){
      m_skillWeights = skillWeights;
   }//setSkillWeights

   public void setInRange(int agentID, boolean value){
      int index = m_agentsCommitted.indexOf(agentID);
      if(index >= 0){
         boolean inRange =
((Boolean)getItem(index,m_agentsInRange)).booleanValue();
         if(!inRange){
            setItem(index,m_agentsInRange,value);
         }//if
      }else{
         writeLog("Bad Index in setInRange -> AgentID=" + agentID + " index=" +
index);
      }//else
   }//setInRange

   private void writeLog(String line){
      parent.writeLog(line);
   }//writeLog

   public int getID(){
      return m_id;
   }//getID

   public int getX(){
      return m_x;
   }//getX

   public int getY(){
      return m_y;
   }//getY

   public float getUtility(){
      float utility = 0.0f;
```

```
      for(int i = 0; i < getLength(m_skillPayoffs); i++){
         float payoff = ((Float)getItem(i,m_skillPayoffs)).floatValue();
         utility = utility + payoff;
      }//for
      return utility;
   }//getUtility

   public float getPayoff(int skillID){
      int index = m_skillsRequired.indexOf(skillID);
      float payoff = 0.0f;
      if(index >= 0){
         payoff = ((Float)getItem(index,m_skillPayoffs)).floatValue();
      }else{
         writeLog("Bad index in getPayoff -> skillID=" + skillID + " index=" +
index);
      }//else
      return payoff;
   }//getPayoff

   public int getState(){
      return m_state;
   }//getState

   public int agentsNeeded(){
      int agentsNeeded = 0;
      for(int j = 0; j < getLength(m_agentsCommitted); j++){
         int taskAgentID = ((Integer)getItem(j,m_agentsCommitted)).intValue();
         if(taskAgentID < 0){
            agentsNeeded++;
         }//if
      }//for
      return agentsNeeded;
   }//agentsNeeded

   /* This function returns true if the task still needs skill
*/
   public boolean isNeeded(int skill){
      if(m_useCommandSkills){
         int commandSkillRequired =
((Integer)getItem(0,m_skillsRequired)).intValue();
         int commandAgentCommitted =
((Integer)getItem(0,m_agentsCommitted)).intValue();
         //Skill 0 has to be filled first since it is a command and control
skill
         if(skill != 0 && commandSkillRequired == 0 && commandAgentCommitted <
0){
            return false;
         }//if
      }//if

      for(int i = 0; i < getLength(m_skillsRequired); i++){
         int neededSkill = ((Integer)getItem(i,m_skillsRequired)).intValue();
         int agentId = ((Integer)getItem(i,m_agentsCommitted)).intValue();
         if(agentId < 0 && skill == neededSkill){
            return true;
         }//if
      }//for
      return false;
   }//isNeeded

   public int findSkillIndex(int skill){
      for(int i = 0; i < getLength(m_agentsCommitted); i++){
         int neededSkill = ((Integer)getItem(i,m_skillsRequired)).intValue();
```

```
         int agentId = ((Integer)getItem(i,m_agentsCommitted)).intValue();
         if(agentId < 0 && skill == neededSkill){
            return i;
         }//if
      }//for
      return -1;
   }// findSkillIndex

   /* This function returns true if all skills of task have been assigned
*/
   public boolean teamFormed(){
      boolean formed = true;
      for(int i = 0; i < getLength(m_skillsRequired) && formed; i++){
         int agentID = ((Integer)getItem(i,m_agentsCommitted)).intValue();
         if(agentID < 0){
            formed = false;
         }//if
         boolean inRange =
((Boolean)getItem(i,m_agentsInRange)).booleanValue();
         if(!inRange && m_useTaskRange){
            formed = false;
         }//if
      }//for
      return formed;
   }//teamFormed

   public int getAgentsInRangeCount(){
      int count = 0;
      if(!m_useTaskRange){
         return getLength(m_skillsRequired);
      }//if
      for(int i = 0; i < getLength(m_skillsRequired); i++){
         boolean inRange =
((Boolean)getItem(i,m_agentsInRange)).booleanValue();
         if(inRange){
            count++;
         }//if
      }//for
      return count;
   }//getAgentsInRangeCount

   public int getMaxAdvertisedTime(){
      return m_maxAdvertisedTime;
   }//getMaxAdvertisedTime

   public int getDuration(){
      return m_duration;
   }//getDuration

   public Vector getSkillsRequired(){
      return m_skillsRequired;
   }//getSkillsRequired

   public Vector getAgentsCommitted(){
      return m_agentsCommitted;
   }//getAgentsCommitted

   public void addAgentCommitted(int id, int skill){
      setItem(skill,m_agentsCommitted,new Integer(id));
      m_agentsCommittedCt++;
   }//addAgentCommitted

   public void removeAgentCommitted(int id){
```

```
      for(int i = 0; i < getLength(m_agentsCommitted); i++){
         int committedID = ((Integer)getItem(i,m_agentsCommitted)).intValue();
         if(committedID == id){
            setItem(i,m_agentsCommitted,new Integer(-1));
            m_agentsCommittedCt--;

            setItem(i,m_agentsInRange,new Boolean(!m_useTaskRange));
         }//if
      }//for
   }//addAgentCommitted

   public Color getColor(){
      return m_color;
   }//getColor

   public String getSkillsString(){
      String result = "";
      for(int i = 0; i < getLength(m_skillsRequired); i++){
         int skill = ((Integer)getItem(i,m_skillsRequired)).intValue();
         result = result + " " + skill;
      }//for
      return result;
   }//getSkillsString

   public String getAgentsString(){
      String result = "";
      for(int i = 0; i < getLength(m_agentsCommitted); i++){
         int agent = ((Integer)getItem(i,m_agentsCommitted)).intValue();
         result = result + " " + agent;
      }//for
      return result;
   }//getAgentsString

   public String getTeamString(){
      return m_teamString;
   }//getTaskString

   public boolean skillRequirementsFilled(){
      for(int i = 0; i < getLength(m_agentsCommitted); i++){
         int agentID = ((Integer)getItem(i,m_agentsCommitted)).intValue();
         if(agentID < 0){
            return false;
         }//if
         boolean inRange =
((Boolean)getItem(i,m_agentsInRange)).booleanValue();
         if(!inRange && m_useTaskRange){
            return false;
         }//if
      }//for
      return true;
   }//skillRequirementsFilled

   public int unFilledSkillsCount(){
      int count = 0;
      for(int i = 0; i < getLength(m_agentsCommitted); i++){
         int agentID = ((Integer)getItem(i,m_agentsCommitted)).intValue();
         if(agentID < 0){
            count++;
         }//if
      }//for
      return count;
   }//unFilledSkillsCount
```

```
   private int outOfRangeCount(){
      int count = 0;
      for(int i = 0; i < getLength(m_agentsInRange); i++){
         boolean inRange =
((Boolean)getItem(i,m_agentsInRange)).booleanValue();
         if(!inRange){
            count++;
         }//if
      }//for
      return count;
   }//outOfRangeCount

   public void updateAdvertised(){
      if(m_maxAdvertisedTime > 0){
         m_maxAdvertisedTime--;
         if(m_maxAdvertisedTime <= 0){
            if(m_state == C_PENDING){
               if(unFilledSkillsCount() == 0){
                  int numOutOfRange = outOfRangeCount();
                  if(numOutOfRange > 0){
                     m_state = C_DISTANCE_FAIL;
                  }//if
               }else{
                  m_state = C_SKILLS_FAIL;
               }//else
            }//if
            String results[] = {"Task Pending", "Skills Unfilled", "Agents
didn't reach task", "Task completed"};
            writeLog(" Task Expired "+ toString() + " result=" +
results[m_state]);
         }//if
      }//if
   }//updateAdvertised

   public float getAgentsCommittedRatio(){
      return  m_agentsCommittedCt/getLength(m_agentsCommitted);
   }//getAgentsCommittedRatio

   public int [] getSkillTally(){
      int [] tally = new int [m_skillMax];
      for(int j = 0; j < m_skillMax; j++){
         tally[j]=0;
      }//for
      for(int i = 0 ; i < getLength(m_skillsRequired); i++){
         int agentCommitted =
((Integer)getItem(i,m_agentsCommitted)).intValue();
         if(agentCommitted < 0){
            tally[((Integer)getItem(i,m_skillsRequired)).intValue()]++;
         }//if
      }//for
      return tally;
   }//getSkillTally

   public boolean isExpired(){
      if(m_maxAdvertisedTime <= 0){
         return true;
      }//if
      return false;
   }//isExpired

   public boolean isTaken(){
      if(isExpired() || m_agentsCommittedCt > 0){
         return true;
```

```
   }//if
   return false;
}//isTaken

public void setSuccessful(){
   m_state = C_TASK_COMPLETE;
}//setSuccessful

public void setClusterIndex(int clusterIndex){
   m_clusterIndex = clusterIndex;
}//setClusterIndex

public int getClusterIndex(){
   return m_clusterIndex;
}//getClusterIndex

public boolean taskImpossible(Hashtable systemSkillHistogram){
   Hashtable taskSkillHistogram = new Hashtable();
   for(int i = 0; i < getLength(m_skillsRequired); i++){
      int skill = (Integer)getItem(i,m_skillsRequired);
      String skillKey = String.valueOf(skill);
      int count = 1;
      if(taskSkillHistogram.containsKey(skillKey)){
         int countSkill = (Integer)taskSkillHistogram.get(skillKey);
         count += countSkill;
      }//if
      taskSkillHistogram.put(skillKey, count);
   }//for

   for(int i = 0; i < getLength(m_skillsRequired); i++){
      String skillKey = String.valueOf(getItem(i,m_skillsRequired));
      int taskSkillCount = (Integer)taskSkillHistogram.get(skillKey);
      int systemSkillCount = (Integer)systemSkillHistogram.get(skillKey);
      if(taskSkillCount > systemSkillCount){
         return true;
      }//if
   }//for

   return false;
}//taskImpossible

public boolean taskImpossible(SkillPriorityList systemSkillHistogram){
   Hashtable taskSkillHistogram = new Hashtable();
   for(int i = 0; i < getLength(m_skillsRequired); i++){
      int skillID = (Integer)getItem(i,m_skillsRequired);
      String skillKey = String.valueOf(skillID);
      int count = 1;
      if(taskSkillHistogram.containsKey(skillKey)){
         int countSkill = (Integer)taskSkillHistogram.get(skillKey);
         count += countSkill;
      }//if
      taskSkillHistogram.put(skillKey, count);
   }//for

   for(int i = 0; i < getLength(m_skillsRequired); i++){
      int skillID = (Integer)getItem(i,m_skillsRequired);
      String skillKey = String.valueOf(skillID);
      int taskSkillCount = (Integer)taskSkillHistogram.get(skillKey);

      SkillPriorityItem item = systemSkillHistogram.get(skillID);
      int systemSkillCount = item.availableAgentCount();
      if(taskSkillCount > systemSkillCount){
         return true;
```

```
         }//if
      }//for

      return false;
   }//taskImpossible

   public boolean taskImpossible(Vector<Integer> neighborSkillCount){
      Hashtable taskSkillHistogram = new Hashtable();
      for(int i = 0; i < getLength(m_skillsRequired); i++){
         int skillID = (Integer)getItem(i,m_skillsRequired);
         String skillKey = String.valueOf(skillID);
         int count = 1;
         if(taskSkillHistogram.containsKey(skillKey)){
            int countSkill = (Integer)taskSkillHistogram.get(skillKey);
            count += countSkill;
         }//if
         taskSkillHistogram.put(skillKey, count);
      }//for

      for(int i = 0; i < getLength(m_skillsRequired); i++){
         int skillID = (Integer)getItem(i,m_skillsRequired);
         String skillKey = String.valueOf(skillID);
         int taskSkillCount = (Integer)taskSkillHistogram.get(skillKey);

         int systemSkillCount = neighborSkillCount.elementAt(skillID);
         if(taskSkillCount > systemSkillCount){
            return true;
         }//if
      }//for

      return false;
   }//taskImpossible

   public void clearAgentsCommitted(){
      for(int i = 0; i < getLength(m_agentsCommitted); i++){
         setItem(i,m_agentsCommitted,-1);
      }//for
   }//clearAgentsCommitted

   public Vector<Integer> countUnfilledSkills(){
      Vector<Integer> unfilledSkillsCount = new Vector();
      for(int i = 0; i < m_skillMax; i++){
         unfilledSkillsCount.add(0);
      }//for
      for(int i = 0; i < getLength(m_skillsRequired); i++){
         int skillID = ((Integer)getItem(i,m_skillsRequired)).intValue();
         int agentId = ((Integer)getItem(i,m_agentsCommitted)).intValue();
         if(agentId < 0){
            int count = unfilledSkillsCount.elementAt(skillID);
            unfilledSkillsCount.setElementAt(count+1,skillID);
         }//if
      }//for
      return unfilledSkillsCount;
   }//countUnfilledSkills

   public Vector<Integer> countUnfilledSkills(Vector<Integer>
unfilledSkillsCount){
      for(int i = 0; i < getLength(m_skillsRequired); i++){
         int skillID = ((Integer)getItem(i,m_skillsRequired)).intValue();
         int agentId = ((Integer)getItem(i,m_agentsCommitted)).intValue();
         if(agentId < 0){
            int count = unfilledSkillsCount.elementAt(skillID);
            unfilledSkillsCount.setElementAt(count+1,skillID);
```

```
         }//if
      }//for
      return unfilledSkillsCount;
   }//countUnfilledSkills

   public String toString(){
      return "Task " + m_id+" x=" + m_x + " y=" + m_y
        + "\n  maxAdvertisedTime=" + m_maxAdvertisedTime + " duration="+
m_duration
        + "\n  skillsRequred=" + m_skillsRequired
        + "\n  skillWeights=" + m_skillWeights
        + "\n  agentsCommittedCt=" + m_agentsCommittedCt
        + "\n  agentsInRangeCt=" + getAgentsInRangeCount()
        + "\n  agentsCommitted=" + m_agentsCommitted
        + "\n  agentsInRange=" + m_agentsInRange;
   }//toString
}//class AONTask

--------------------------------AONTask.java-------------------------------------
-

-----------------------------AONTaskList.java---------------------------------
-

import java.util.Vector;

/**
 * AONTaskList Class
 *
 *
 */
public class AONTaskList{
   int m_firstCurrentIndex;  /*
*/
   Vector<AONTask> m_tasks;

   /* constructor
*/
   public AONTaskList(){
      m_firstCurrentIndex = 0;
      m_tasks = new Vector();
   }//AONTaskList

   public void add(AONTask task){
      m_tasks.add(task);
   }//add

   public AONTask getItem(int index){
      if(index >= 0){
         return m_tasks.elementAt(index);
      }else{
         return null;
      }//else
   }//getItem

   public int getFirstCurrentIndex(){
      return m_firstCurrentIndex;
   }//getFirstCurrentIndex

   public void updateFirstCurrentIndex(){
      int i = 0;
      boolean noCurrent = true;
      for(i = m_firstCurrentIndex; i < m_tasks.size() && noCurrent; i++){
```

```
         AONTask task = m_tasks.elementAt(i);
         if(!task.isExpired()){
            noCurrent = false;
         }//if
      }//for
      if(i > 0){
         m_firstCurrentIndex = i - 1;
      }else{
         m_firstCurrentIndex = 0;
      }//else
   }//updateFirstCurrentIndex

   public int getLength(){
      return m_tasks.size();
   }//getLength
}//AONTaskList

------------------------------AONTaskList.java----------------------------------
-

------------------------------AONXMLReader.java---------------------------------
-

import java.io.*;
import java.util.*;

/**
 * AONXMLReader Class
 *
 *
 */
public class AONXMLReader implements DocHandler{
   private XMLElement m_Root;/*
*/
   private XMLElement m_CurrentElement;
                              /*
*/

   /* constructor
*/
   public AONXMLReader(){
      super();
   }//constructor

   public boolean openXML(String filename){
      boolean success = true;
      try{
         FileReader r = new FileReader(filename);
         QDParser.parse(this,r);
      }catch(Exception ex){
         System.out.println(ex.toString());
         success = false;
      }//catch
      return success;
   }//openXML

   public void startElement(String name,Hashtable h){
      XMLElement last = m_CurrentElement;
      m_CurrentElement = new XMLElement(name, last);

      Enumeration e = h.keys();
      while(e.hasMoreElements()){
         String key, value;
```

```
            key = (String)e.nextElement();
            value = (String)h.get(key);
            m_CurrentElement.addAttribute(key,value);
         }//while

         if(m_Root == null){
            m_Root = m_CurrentElement;
         }else{
            if(last != null){
               last.addElement(name,m_CurrentElement);
            }//if
         }//else
      }//startElement

      public void endElement(String tag){
         m_CurrentElement = m_CurrentElement.getParent();
      }//endElement

      public void text(String str) throws Exception{};

      public void startDocument(){
         //System.out.println("Start document");
      }//startDocument

      public void endDocument(){
         //System.out.println("End document");
      }//endDocument

      public XMLElement getRoot(){
         return m_Root;
      }//getRoot

      public void setRoot(XMLElement r){
         m_Root = r;
      }//setRoot

      public boolean writeXML(String filename){
         boolean success = true;
         try{
            BufferedWriter fileOut = new BufferedWriter(new FileWriter(filename));
            if(!writeElement(m_Root, fileOut, 0)){
               return false;
            }//if
            fileOut.close();
         }catch(Exception ex){
            System.out.println(ex.toString());
            success = false;
         }//catch
         return success;
      }//writeXML

      private boolean writeElement(XMLElement xml, BufferedWriter fileOut, int
level){
         boolean success = true;
         try{
            String attributes = "";
            for(int i = 0; i < xml.getNumAttributes(); i++){
               attributes = attributes + " " + xml.getAttributeName(i) + "=\"" +
xml.getAttribute(i) + "\"";
            }//for
            String header = "";
            if(attributes != ""){
               header = "<" + xml.getName() + attributes + ">";
```

```
         }else{
            header = "<" + xml.getName() + ">";
         }//else
         fileOut.write(getTab(level) + header);
         fileOut.newLine();
         for(int i = 0; i < xml.getNumElements(); i++){
            writeElement(xml.getElement(i), fileOut, level+1);
         }//for
         String footer = "</" + xml.getName() + ">";
         fileOut.write(getTab(level) + footer);
         fileOut.newLine();
      }catch(Exception ex){
         success = false;
      }//catch
      return success;
   }//writeElement

   private String getTab(int level){
      String tab = "";
      for(int i = 0; i < level; i++){
         //tab = tab + "\t";
         tab = tab + "  ";
      }//for
      return tab;
   }//getTab
}//class AONXMLReader
```

----------AONXMLReader.java----------
-

----------CSVCache.java----------
-

```
import java.util.Vector;

/**
 * CSVCache Class
 *
 *
 */
public class CSVCache{
   int m_cacheStartRowIndex; /*
*/
   Vector m_cache;           /*
*/
   /* constructor
*/
   public CSVCache(){
      m_cacheStartRowIndex = 0;
      m_cache = new Vector();
   }//constructor

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
```

```
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   public void addToCache(int rowIndex, String line){
      if(getLength(m_cache) == 0){
         m_cacheStartRowIndex = rowIndex;
      }//if
      m_cache.add(line);
   }//addToCache

   public String checkCache(int rowIndex){
      int index = rowIndex - m_cacheStartRowIndex;
//       System.out.println("Cache: " + m_cache);
//       System.out.println("checkCache index: " + index);
      if(index >= 0 && index < getLength()){
         return (String)getItem(index,m_cache);
      }//if
      return null;
   }//checkCache

   public int getLength(){
      return getLength(m_cache);
   }//getLength

   public void clear(){
      m_cache.clear();
   }//clear
}//class CSVCache

------------------------------CSVCache.java-------------------------------------
-

------------------------------CSVFile.java--------------------------------------
-

import java.util.Vector;
import java.util.StringTokenizer;
import java.io.IOException;
import java.io.BufferedWriter;
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.FileReader;
import java.io.File;
```

```
/**
 * CSVFile Class
 *
 *
 */
public class CSVFile{
   int m_lineCount;          /*
*/
   Vector m_lines;           /*
*/
   File m_tempFile;          /*
*/

   int m_cacheLinesCount = 1000;
   CSVCache m_lineCache;     /*
*/

   float m_maxValue;         /*
*/
   float m_minValue;         /*
*/

   /* constructor
*/
   public CSVFile(){
      m_lines = new Vector();
      m_lineCache = new CSVCache();
      m_maxValue = -1.0f;
      m_minValue = 999999999.0f;
      try{
         m_tempFile = File.createTempFile("TempCSVFile",".csv",null);
         m_tempFile.deleteOnExit();
         //System.out.println(m_tempFile.getAbsolutePath());
      }catch(IOException ex){
         System.out.println("Error creating temp csv file " +
m_tempFile.getAbsolutePath() + ": " + ex);
      }//catch
   }//CSVFile

   /* constructor
*/
   public CSVFile(String filename){
      m_lines = new Vector();
      m_lineCache = new CSVCache();
      m_maxValue = -1.0f;
      m_minValue = 999999999.0f;
      try{
         m_tempFile = File.createTempFile("TempCSVFile",".csv",null);
         m_tempFile.deleteOnExit();
         //System.out.println(m_tempFile.getAbsolutePath());
         open(filename);
      }catch(IOException ex){
         System.out.println("Error creating temp csv file " +
m_tempFile.getAbsolutePath() + ": " + ex);
      }//catch
   }//CSVFile

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
```

```
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   protected void close(){
      m_tempFile.delete();
   }//finalize

   public boolean open(String filename){
      try{
         BufferedReader in = new BufferedReader(new FileReader(filename));
         String line = in.readLine();
         while(line != null){
            //System.out.println("Input line: " + line);
            updateMaxMinValues(line);
            writeLine(line);
            if(m_lineCache.getLength() < m_cacheLinesCount){
               m_lineCache.addToCache(m_lineCount,line);
            }//if
            //pauseTest("Line " + m_lineCount + " in file " + filename +
"...");
            m_lineCount++;
            line = in.readLine();
         }//while
         //System.out.println("Closing file...");
         in.close();
      }catch(IOException ex){
         System.out.println("Error reading file " + filename + ": " +
ex.getMessage());
         return false;
      }//catch
      return true;
   }//open

   private void writeLine(String line){
      if(line != null){
         try{
```

```
            BufferedWriter out = new BufferedWriter(new
FileWriter(m_tempFile.getAbsolutePath(),true));
            out.write(line);
            out.newLine();
            out.close();
            //System.out.println("Wrote line to file: " +
m_tempFile.getAbsolutePath());
         }catch(IOException ex){
            System.out.println("Error writing line to file " +
m_tempFile.getAbsolutePath() + ": " + ex.getMessage());
         }//catch
      }//if
   }//writeLine

   public boolean save(String filename){
      try{
         int index = 0;
         BufferedReader in = new BufferedReader(new FileReader(m_tempFile));
         BufferedWriter out = new BufferedWriter(new FileWriter(filename));
         String line = in.readLine();
         while(line != null){
            //System.out.println("Input line: " + line);
            updateMaxMinValues(line);
            out.write(line);
            out.newLine();
            index++;
            //pauseTest("Line " + index + " in file " + filename + "...");
            line = in.readLine();
         }//while
         in.close();
         out.close();
         System.out.println("Wrote file: " + filename);
      }catch(IOException ex){
         System.out.println("Error writing file " + filename + ": " +
ex.getMessage());
         return false;
      }//catch
      return true;
   }//save

   public void addRow(Vector values){
      String line = "";
      for(int i = 0; i < getLength(values); i++){
         if(line.compareTo("") == 0){
            line = (String)getItem(i,values);
            if(line == null){
               line = " ";
            }//if
         }else{
            line = line + "," + (String)getItem(i,values);
         }//else
      }//for
      writeLine(line);
      m_lineCount++;
   }//addRow

   public void addRow(String values[]){
      String line = "";
      for(int i = 0; i < values.length; i++){
         if(line.compareTo("") == 0){
            line = values[i];
            if(line == null){
               line = " ";
```

```
            }//if
         }else{
            line = line + "," + values[i];
         }//else
      }//for
      writeLine(line);
      m_lineCount++;
   }//addRow

   private void parseLine(int lineIndex, String line){
      int index = 0;
      Vector values = new Vector();
      String item = "";
      StringTokenizer splitString = new StringTokenizer(line,",");
      while(splitString.hasMoreTokens()){
         item = splitString.nextToken();
         //System.out.print(item + " ");
         values.add(item);
         index++;
      }//while
      m_lines.add(values);
      //AONReportCatagory c = (AONReportCatagory)getItem(0,m_catagories);
      //System.out.println("AONReportCatagory 1: " + c.toString());
      //System.out.println("\nPerformance: " +
m_performanceLineValues.toString());
      //System.out.println("\nEfficiency: " +
m_efficiencyLineValues.toString());
   }//parseLine

   public String getItem(int row, int col){
      //System.out.println("CSVFile.getItem: Begin");
      String returnValue = null;
      String cacheLine = m_lineCache.checkCache(row);
      if(cacheLine != null){
         //System.out.println("Look in cache");
         int tokenIndex = 0;
         StringTokenizer splitString = new StringTokenizer(cacheLine,",");
         while(splitString.hasMoreTokens()){
            String value = splitString.nextToken();
            if(tokenIndex == col){
               returnValue = value;
               //System.out.println("getItem hit cache...");
            }//if
            tokenIndex++;
         }//while
      }else{
         //System.out.println("Read lines from csv file");
         try{
            int index = 0;
            boolean cacheFilling = false;
            BufferedReader in = new BufferedReader(new FileReader(m_tempFile));
            String line = in.readLine();
            while(line != null && (returnValue == null || cacheFilling)){
               //System.out.println("Input line: " + line);
               if(index == row){
                  int tokenIndex = 0;
                  StringTokenizer splitString = new StringTokenizer(line,",");
                  while(splitString.hasMoreTokens()){
                     String value = splitString.nextToken();
                     if(tokenIndex == col){
                        m_lineCache.clear();
                        cacheFilling = true;
                        returnValue = value;
```

```
                        //System.out.println("getItem hit file...");
                     }//if
                     tokenIndex++;
                  }//while
               }//if
               m_lineCache.addToCache(index,line);
               if(m_lineCache.getLength() == m_cacheLinesCount){
                  cacheFilling = false;
               }//if
               //pauseTest("Line " + index + " in file " +
m_tempFile.getAbsolutePath() + "...");
               index++;
               line = in.readLine();
            }//while
            in.close();
         }catch(IOException ex){
            System.out.println("Error reading file " +
m_tempFile.getAbsolutePath() + ": " + ex.getMessage());
         }//catch
      }//else
      //System.out.println("CSVFile.getItem: End");
      return returnValue;
   }//getItem

   public void setItem(int row, int col, String newValue){
      try{
         int index = 0;
         BufferedReader in = new BufferedReader(new FileReader(m_tempFile));
         String line = in.readLine();
         while(line != null){
            //System.out.println("Input line: " + line);
            if(index == row){
               int tokenIndex = 0;
               Vector values = new Vector();
               StringTokenizer splitString = new StringTokenizer(line,",");
               while(splitString.hasMoreTokens()){
                  String value;
                              /*
*/
                  if(tokenIndex == col){
                     value = newValue;
                  }else{
                     value = splitString.nextToken();
                  }//else
                  values.add(value);
                  tokenIndex++;
               }//while
            }//if
            index++;
            //pauseTest("Line " + index + " in file " + filename + "...");
            line = in.readLine();
         }//while
         in.close();
      }catch(IOException ex){
         System.out.println("Error reading file " +
m_tempFile.getAbsolutePath() + ": " + ex.getMessage());
      }//catch
   }//setItem

   public int getLineCount(){
      return m_lineCount;
   }//getLineCount
```

```
   public float getMaxValue(){
      return m_maxValue;
   }//getMaxFloatValue

   public float getMinValue(){
      return m_minValue;
   }//getMinValue

   private void updateMaxMinValues(String line){
      int index = 0;
      Vector values = new Vector();
      String item = "";
      StringTokenizer splitString = new StringTokenizer(line,",");
      while(splitString.hasMoreTokens()){
         item = splitString.nextToken();
         if(index > 0){
            float tempFloatValue = -1.0f;
            try{
               tempFloatValue = Float.parseFloat(item);
               if(tempFloatValue > m_maxValue){
                  m_maxValue = tempFloatValue;
               }//if
               if(tempFloatValue < m_minValue){
                  m_minValue = tempFloatValue;
               }//if
            }catch(Exception ex){
            }//catch
         }//if
         index++;
      }//while
   }//updateMaxMinValues

   public void pauseTest(String message){
      try{
          char input = ' ';
          byte temp[] = new byte[1];
          System.out.print("\n" + message + "\nPress <enter> to continue...");
          while(input != '\n'){
            System.in.read(temp);
            input = (char)temp[0];
          }//while
      }catch(IOException ex){
      }//catch
   }//pauseTest
}//CSVFile

------------------------------CSVFile.java-------------------------------------
-

--------------------------DiversityAgent.java---------------------------------
-

import java.util.*;

/**
 * DiversityAgent Class
 *   This is an agent that seeks to rewire to agents with different skills
 *     than the skills possessed by the agent
 */
public class DiversityAgent extends AgentX{
   /* constructor
*/
   public DiversityAgent(TestAON p,int id, int x, int y, Vector agents){
```

```
      super(p,id,x,y,"Diversity",agents);
   }//constructor

   /* This function is used to find the best agent recommendation for a new
*/
   /*   neighbor by doing a recursive search of the network until depth has
*/
   /*   reached 0. The best agent recommendation is returned back to the
*/
   /*   agent who originated the request for agent recommendations
*/
   public AgentRecommendation recommendNeighbor(int depth, Vector
agentsVisited, AgentX sourceAgent){
      int bestAgentID = -1;
      float bestCompareValue = -1.0f;
      depth--;
      agentsVisited.add(m_id);
      if(depth <= 0){
         if(!sourceAgent.hasNeighbor(m_id) && m_id != sourceAgent.getID()){
            bestAgentID = m_id;
            bestCompareValue = getSkillComparison(sourceAgent.getSkills());
         }//if
      }else{
         for(int i = 0; i < m_neighbors.getLength(); i++){
            int id = m_neighbors.getItem(i).agentID();
            if(!contains(id,agentsVisited)){
               AgentX neighbor = (AgentX)getAgent(id);
               AgentRecommendation agentRecommendation =
neighbor.recommendNeighbor(depth,agentsVisited,sourceAgent);
               if(agentRecommendation.getID() >= 0){
                  int agentDegree =
getAgent(agentRecommendation.getID()).getDegree();
                  if((int)agentRecommendation.getValue() > bestCompareValue &&
agentDegree < m_maxDegree){
                     bestAgentID = agentRecommendation.getID();
                     bestCompareValue = agentRecommendation.getValue();
                  }//if
               }//if
            }//if
         }//for
         if(!sourceAgent.hasNeighbor(m_id) && m_id != sourceAgent.getID()){
            if(bestAgentID == -1){
               bestAgentID = m_id;
               bestCompareValue = getSkillComparison(sourceAgent.getSkills());
            }else{
               float personalComparison =
getSkillComparison(sourceAgent.getSkills());
               if(personalComparison > bestCompareValue){
                  bestAgentID = m_id;
                  bestCompareValue = personalComparison;
               }//if
            }//else
         }//if
      }//else
      return new AgentRecommendation(bestAgentID,bestCompareValue);
   }//recommendNeighbor

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "DiversityAgent\n" + super.toString() + "\n";
      return value;
   }//toString
```

```
}//class DiversityAgent

--------------------------DiversityAgent.java--------------------------------
-

----------------------DiversityImpatientAgent.java---------------------------
-

import java.util.*;

/**
 * DiversityImpatientAgent Class
 *   This is an agent that seeks to rewire to agents with different skills
 *     than the skills possessed by the agent
 *   -This agent is 'impatient' because it builds off the Egalitarian agent
 *     by choosing to drop its commitment to a task when the skills left to be
 *     filled in the task exceed the number of neighboring agents who are not
 *     committed to a task
 */
public class DiversityImpatientAgent extends DiversityAgent {
   /* constructor
*/
   public DiversityImpatientAgent(TestAON p,int id, int x, int y, Vector
agents){
      super(p,id,x,y,agents);
      m_type = "DiversityImpatient";
   }//constructor

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      boolean dropTask = false;
      if(getUnCommittedNeighborCount() < task.unFilledSkillsCount()){
         dropTask = true;
      }//if
      return dropTask;
   }//chooseToDropTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "DiversityImpatientAgent\n" + super.toString() + "\n";
      return value;
   }//toString
}//class DiversityImpatientAgent

----------------------DiversityImpatientAgent.java---------------------------
-

-------------------------DiversityStratAgent.java-----------------------------
-

import java.util.*;

/**
 * DiversityStratAgent Class
 *   This is an agent that seeks to rewire to agents with different skills
 *     than the skills possessed by the agent
 *   -This agent is 'strategic' because it builds off the Egalitarian agent
 *    by choosing tasks to join and tasks to propose that are the best match
 *     than the skills possessed by the agent
```

```
 */
public class DiversityStratAgent extends DiversityAgent {
   /* constructor
*/
   public DiversityStratAgent(TestAON p,int id, int x, int y, Vector agents){
      super(p,id,x,y,agents);
      m_type = "DiversityStrat";
   }//constructor

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
       m_edgesChanged = 0;
       writeLog(m_id + " updateUncommitted:");
       // Look at all tasks which your immediate neighbor has committed to
       // Select the one which is best based on (1) one with more committed
agents
       // (2) one for which your neighbors have the most chance of satisfying
       AONTask taskJoin = pickNeighborTask(tasks);
       if(taskJoin != null){
          acceptTask(taskJoin);
          m_state = C_COMMITTED;
          m_numberOfTeamsJoined++;
          m_committedTimeRemaining = m_maxCommittedTime;
          m_lastWorkCatagory = C_JOINED;
         return;
      }//if

      float random = rand.nextFloat();
      // WaitPercent can be fixed or dynamic (based on success with current
strategy)
      Boolean wantToWait = (random < m_waitPercent);
      if(chooseToAdapt()){
         rewire();
      }else if(wantToWait){
         return;
      }else{
         //Propose a new task
         // Look at all available tasks which you can do.
         // Select the one which is best based on (1) random choice
         // (2) one which your neighbors have the most chance of satisfying
         AONTask task = pickTask(tasks);
         if(task != null){
            acceptTask(task);
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
            m_lastWorkCatagory = C_PROPOSED;
            m_numberOfTeamsJoined++;
         }//if
      }//else
   }//updateUncommitted

   /* This function returns the best task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickTask(AONTaskList tasks){
      return pickBestTask(tasks);
   }//pickTask

   public AONTask pickNeighborTask(AONTaskList tasks){
      return pickBestNeighborTask(tasks);
   }//pickNeighborTask
```

```
   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "DiversityStratAgent\n" + super.toString() + "\n";
      return value;
   }//toString
}//class DiversityStratAgent

-------------------------DiversityStratAgent.java-----------------------------
-

---------------------DiversityStratImpatientAgent.java-------------------------
-

import java.util.*;

/**
 * DiversityStratImpatientAgent Class
 *   This is an agent that seeks to rewire to agents with different skills
 *     than the skills possessed by the agent
 *   -This agent is 'strategic' because it builds off the Egalitarian agent
 *    by choosing tasks to join and tasks to propose that are the best match
 *    for the skills possessed by the agent's uncommitted neighbors
 *   -This agent is 'impatient' because it builds off the Egalitarian agent
 *     by choosing to drop its commitment to a task when the skills left to be
 *     filled in the task exceed the number of neighboring agents who are not
 *     committed to a task
 */
public class DiversityStratImpatientAgent extends DiversityAgent {
   /* constructor
*/
   public DiversityStratImpatientAgent(TestAON p,int id, int x, int y, Vector
agents){
      super(p,id,x,y,agents);
      m_type = "DiversityStratImpatient";
   }//constructor

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
       m_edgesChanged = 0;
       writeLog(m_id + " updateUncommitted:");
       // Look at all tasks which your immediate neighbor has committed to
       // Select the one which is best based on (1) one with more committed
agents
       // (2) one for which your neighbors have the most chance of satisfying
       AONTask taskJoin = pickNeighborTask(tasks);
       if(taskJoin != null){
          acceptTask(taskJoin);
          m_state = C_COMMITTED;
          m_numberOfTeamsJoined++;
          m_committedTimeRemaining = m_maxCommittedTime;
          m_lastWorkCatagory = C_JOINED;
         return;
      }//if

      float random = rand.nextFloat();
      // WaitPercent can be fixed or dynamic (based on success with current
strategy)
      Boolean wantToWait = (random < m_waitPercent);
      if(chooseToAdapt()){
         rewire();
```

```
      }else if(wantToWait){
         return;
      }else{
         //Propose a new task
         // Look at all available tasks which you can do.
         // Select the one which is best based on (1) random choice
         // (2) one which your neighbors have the most chance of satisfying
         AONTask task = pickTask(tasks);
         if(task != null){
            acceptTask(task);
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
            m_lastWorkCatagory = C_PROPOSED;
            m_numberOfTeamsJoined++;
         }//if
      }//else
   }//updateUncommitted

   /* This function returns the best task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickNeighborTask(AONTaskList tasks){
      return pickBestNeighborTask(tasks);
   }//pickNeighborTask

   /* This function returns the best task which this agent is needed for
*/
   public AONTask pickTask(AONTaskList tasks){
      return pickBestTask(tasks);
   }//pickTask

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      boolean dropTask = false;
      if(getUnCommittedNeighborCount() < task.unFilledSkillsCount()){
         dropTask = true;
      }//if
      return dropTask;
   }//chooseToDropTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "DiversityStratImpatientAgent\n" + super.toString() +
"\n";
      return value;
   }//toString
}//class DiversityStratImpatientAgent

----------DiversityStratImpatientAgent.java----------
-

------------------------------DocHandler.java------------------------------
-

import java.util.*;

/**
 * DocHandler Interface
```

```
 *
 *
 */
public interface DocHandler{
   public void startElement(String tag,Hashtable h) throws Exception;
   public void endElement(String tag) throws Exception;
   public void startDocument() throws Exception;
   public void endDocument() throws Exception;
   public void text(String str) throws Exception;
}//DocHandler

------------------------------DocHandler.java-----------------------------------
-

------------------------------DummyTask.java------------------------------------
-

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.IOException;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.File;
import javax.swing.Timer;
import java.awt.image.*;
import javax.imageio.ImageIO;

/**
 * DummyTask Class
 *
 *
 */
public class DummyTask{
   int []d_skillCt;
   int d_id;                 /*
*/
   int d_skillsNeeded;       /*
*/
   Random rand;              /*
*/

   /* constructor
*/
   public DummyTask(TestAON test,int id, int numberOfSkillsPerTask, int
numberOfSkills){
      d_id = id;
      d_skillsNeeded = numberOfSkillsPerTask;
      rand = new Random (System.currentTimeMillis());
      d_skillCt = new int [ numberOfSkills];
      for(int i = 0; i < numberOfSkillsPerTask; i++){
         int s = rand.nextInt(numberOfSkills);
         d_skillCt[s]++;
      }//for
   }//constructor

   /* constructor
*/
   public DummyTask(DummyTask t){
      d_id = t.d_id;
      d_skillsNeeded = t.d_skillsNeeded;
      d_skillCt = new int [ t.d_skillCt.length];
      for(int i = 0; i < t.d_skillCt.length; i++){
```

```
         d_skillCt[i]= t.d_skillCt[i];
      }//for
   }//constructor

   boolean needSkill(int skill){
      if(skill >= 0){
         return d_skillCt[skill]>0;
      }else{
         return false;
      }//else
   }//needSkill

   void provideSkill(int skill){
      d_skillsNeeded--;
      d_skillCt[skill]--;
   }//provideSkill

   void unprovideSkill(int skill){
      d_skillsNeeded++;
      d_skillCt[skill]++;
   }//unprovideSkill

   boolean satisfied(){
      return (d_skillsNeeded == 0);
   }//satisfied

   public String toString(){
      String res = "DummyTask " + d_id + "[";
      for (int i=0; i < d_skillCt.length;i++){
         res+= d_skillCt[i] + " ";
      }//for
      return res+ "]";
   }//toString
}//DummyTask

------------------------------DummyTask.java-------------------------------------
-

--------------------------EgalitarianAgent.java--------------------------------
-

import java.util.*;

/**
 * EgalitarianAgent Class
 *   This is an agent that seeks to rewire to agents who have the fewest
 *   network connections
 */
public class EgalitarianAgent extends AgentX{
   /* constructor
*/
   public EgalitarianAgent(TestAON p,int id, int x, int y, Vector agents){
      super(p,id,x,y,"Egalitarian",agents);
   }//constructor

   /* This function returns a neighbor agent id as a neighbor to be
*/
   /*   removed
*/
   public int pickRemoveNeighbor(){
      float worsePerf = 100.0f;
      int worseAgent = -1;
      // Find the worst performance NOT counting the neighbor you just added
```

```
      for(int i = 0; i < m_neighbors.getLength()-1; i++){
         int id = m_neighbors.getItem(i).agentID();
         AgentX agent = (AgentX)getAgent(id);
         float perf = agent.getPerformance();
         if(perf < worsePerf){
            worsePerf = perf;
            worseAgent = id;
         }//if
      }//for

      return worseAgent;
   }//pickRemoveNeighbor

   /* This function returns whether or not the agent has decided to adapt its
*/
   /*   network connections to its neighbors
*/
   public boolean chooseToAdapt(){
      boolean adapt = false;
      float num = (float)m_adaptationRate/getAgentCount();
      float random = rand.nextFloat();
      if(random < num && m_adaptNetwork){
         adapt = true;
      }//if
      return adapt;
   }//chooseToAdapt

   /* This function is used to find the best agent recommendation for a new
*/
   /*   neighbor by doing a recursive search of the network until depth has
*/
   /*   reached 0. The best agent recommendation is returned back to the
*/
   /*   agent who originated the request for agent recommendations
*/
   public AgentRecommendation recommendNeighbor(int depth, Vector
agentsVisited, AgentX sourceAgent){
      int bestAgentID = -1, bestAgentDegree = 20;
      depth--;
      agentsVisited.add(m_id);
      if(depth <= 0){
         if(!sourceAgent.hasNeighbor(m_id) && m_id != sourceAgent.getID()){
            bestAgentID = m_id;
            bestAgentDegree = m_neighbors.getLength();
         }//if
      }else{
         for(int i = 0; i < m_neighbors.getLength(); i++){
            int id = m_neighbors.getItem(i).agentID();
            if(!contains(id,agentsVisited)){
               AgentX neighbor = (AgentX)getAgent(id);
               AgentRecommendation agentRecommendation =
neighbor.recommendNeighbor(depth,agentsVisited,sourceAgent);
               if(agentRecommendation.getID() >= 0){
                  if((int)agentRecommendation.getValue() < bestAgentDegree &&
(int)agentRecommendation.getValue() < m_maxDegree){
                     bestAgentID = agentRecommendation.getID();
                     bestAgentDegree = (int)agentRecommendation.getValue();
                  }//if
               }//if
            }//if
         }//for
         if(!sourceAgent.hasNeighbor(m_id) && m_id != sourceAgent.getID()){
            if(bestAgentID == -1){
```

```
               bestAgentID = m_id;
               bestAgentDegree = m_neighbors.getLength();
            }else{
               if(m_neighbors.getLength() < bestAgentDegree){
                  bestAgentID = m_id;
                  bestAgentDegree = m_neighbors.getLength();
               }//if
            }//else
         }//if
      }//else
      return new AgentRecommendation(bestAgentID,bestAgentDegree);
   }//recommendNeighbor

   /* This function returns the agent id of the most highly recommended
*/
   /*   agent who may become this agent's new neighbor
*/
   public int pickNewNeighbor(){
      AgentRecommendation recommended =
recommendNeighbor(m_communicationDepth+1,new Vector(),this);
      return recommended.getID();
   }//pickNewNeighbor

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "EgalitarianAgent_" + super.toString() + "\n";
      return value;
   }//toString
}//class EgalitarianAgent

-------------------------EgalitarianAgent.java--------------------------------
-

----------------------EgalitarianImpatientAgent.java----------------------------
-

import java.util.*;

/**
 * EgalitarianImpatientAgent Class
 *   This is an agent that seeks to rewire to agents who have the fewest
 *   network connections
 *   -This agent is 'impatient' because it builds off the Egalitarian agent
 *     by choosing to drop its commitment to a task when the skills left to be
 *     filled in the task exceed the number of neighboring agents who are not
 *     committed to a task
 */
public class EgalitarianImpatientAgent extends EgalitarianAgent{
   /* constructor
*/
   public EgalitarianImpatientAgent(TestAON p,int id, int x, int y, Vector
agents){
      super(p,id,x,y,agents);
      m_type = "EgalitarianImpatient";
   }//constructor

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      boolean dropTask = false;
```

```
      if(getUnCommittedNeighborCount() < task.unFilledSkillsCount()){
         dropTask = true;
      }//if
      return dropTask;
   }//chooseToDropTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "EgalitarianImpatientAgent_" + super.toString() + "\n";
      return value;
   }//toString
}//class EgalitarianImpatientAgent
```

----------EgalitarianImpatientAgent.java----------

----------EgalitarianStratAgent.java----------

```
import java.util.*;

/**
 * EgalitarianStratAgent Class
 *   This is an agent that seeks to rewire to agents who have the fewest
 *   network connections
 *   -This agent is 'strategic' because it builds off the Egalitarian agent
 *    by choosing tasks to join and tasks to propose that are the best match
 *    for the skills possessed by the agent's uncommitted neighbors
 */
public class EgalitarianStratAgent extends EgalitarianAgent{
   /* constructor
*/
   public EgalitarianStratAgent(TestAON p,int id, int x, int y, Vector agents){
      super(p,id,x,y,agents);
      m_type = "EgalitarianStrat";
   }//constructor

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
       m_edgesChanged = 0;
       writeLog(m_id + " updateUncommitted:");
       // Look at all tasks which your immediate neighbor has committed to
       // Select the one which is best based on (1) one with more committed
agents
       // (2) one for which your neighbors have the most chance of satisfying
       AONTask taskJoin = pickNeighborTask(tasks);
       if(taskJoin != null){
          acceptTask(taskJoin);
          m_state = C_COMMITTED;
          m_numberOfTeamsJoined++;
          m_committedTimeRemaining = m_maxCommittedTime;
          m_lastWorkCatagory = C_JOINED;
         return;
      }//if

      float random = rand.nextFloat();
      // WaitPercent can be fixed or dynamic (based on success with current
strategy)
      Boolean wantToWait = (random < m_waitPercent);
      if(chooseToAdapt()){
```

```
         rewire();
      }else if(wantToWait){
         return;
      }else{
         //Propose a new task
         // Look at all available tasks which you can do.
         // Select the one which is best based on (1) random choice
         // (2) one which your neighbors have the most chance of satisfying
         AONTask task = pickTask(tasks);
         if(task != null){
            acceptTask(task);
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
            m_lastWorkCatagory = C_PROPOSED;
            m_numberOfTeamsJoined++;
         }//if
      }//else
   }//updateUncommitted

   /* This function returns the best task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickNeighborTask(AONTaskList tasks){
      return pickBestNeighborTask(tasks);
   }//pickNeighborTask

   /* This function returns the best task which this agent is needed for
*/
   public AONTask pickTask(AONTaskList tasks){
      return pickBestTask(tasks);
   }//pickTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "EgalitarianStratAgent_" + super.toString() + "\n";
      return value;
   }//toString
}//class EgalitarianStratAgent

----------------------EgalitarianStratAgent.java-----------------------------
-

------------------EgalitarianStratImpatientAgent.java--------------------------
-

import java.util.*;

/**
 * EgalitarianStratImpatientAgent Class
 *   This is an agent that seeks to rewire to agents who have the fewest
 *   network connections
 *   -This agent is 'strategic' because it builds off the Egalitarian agent
 *    by choosing tasks to join and tasks to propose that are the best match
 *    for the skills possessed by the agent's uncommitted neighbors
 *   -This agent is 'impatient' because it builds off the Egalitarian agent
 *     by choosing to drop its commitment to a task when the skills left to be
 *     filled in the task exceed the number of neighboring agents who are not
 *     committed to a task
 */
public class EgalitarianStratImpatientAgent extends EgalitarianAgent{
```

```
   /* constructor
*/
   public EgalitarianStratImpatientAgent(TestAON p,int id, int x, int y, Vector
agents){
      super(p,id,x,y,agents);
      m_type = "EgalitarianStratImpatient";
   }//constructor

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
       m_edgesChanged = 0;
       writeLog(m_id + " updateUncommitted:");
       // Look at all tasks which your immediate neighbor has committed to
       // Select the one which is best based on (1) one with more committed
agents
       // (2) one for which your neighbors have the most chance of satisfying
       AONTask taskJoin = pickNeighborTask(tasks);
       if(taskJoin != null){
          acceptTask(taskJoin);
          m_state = C_COMMITTED;
          m_numberOfTeamsJoined++;
          m_committedTimeRemaining = m_maxCommittedTime;
          m_lastWorkCatagory = C_JOINED;
         return;
      }//if

      float random = rand.nextFloat();
      // WaitPercent can be fixed or dynamic (based on success with current
strategy)
      Boolean wantToWait = (random < m_waitPercent);
      if(chooseToAdapt()){
         rewire();
      }else if(wantToWait){
         return;
      }else{
         //Propose a new task
         // Look at all available tasks which you can do.
         // Select the one which is best based on (1) random choice
         // (2) one which your neighbors have the most chance of satisfying
         AONTask task = pickTask(tasks);
         if(task != null){
            acceptTask(task);
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
            m_lastWorkCatagory = C_PROPOSED;
            m_numberOfTeamsJoined++;
         }//if
      }//else
   }//updateUncommitted

   /* This function returns the best task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickNeighborTask(AONTaskList tasks){
      return pickBestNeighborTask(tasks);
   }//pickNeighborTask

   /* This function returns the best task which this agent is needed for
*/
   public AONTask pickTask(AONTaskList tasks){
      return pickBestTask(tasks);
```

```
   }//pickTask

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      boolean dropTask = false;
      if(getUnCommittedNeighborCount() < task.unFilledSkillsCount()){
         dropTask = true;
      }//if
      return dropTask;
   }//chooseToDropTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "EgalitarianStratImpatientAgent_" + super.toString() +
"\n";
      return value;
   }//toString
}//class EgalitarianStratImpatientAgent
```

----------EgalitarianStratImpatientAgent.java----------

----------FailedAgentX.java----------

```
import java.awt.*;
import java.util.*;
import java.io.IOException;

/**
 * FailedAgentX Class
 *   This is the clas representing an agent who has failed, basically still
 *   having the same interface, but no functionality behind the interface
 */
public class FailedAgentX extends AgentX{
   AgentX m_agent;           /* The failed agent that is being represented
*/
   int m_oldDegree;          /* The number of neighbors the agent had when
*/
                             /* it failed
*/

   /* Default constructor
*/
   public FailedAgentX(){
      m_agent = new AgentX();
   }//constructor

   /* constructor
*/
   public FailedAgentX(int id, int x, int y, int springX, int springY){
      m_id = id;
      m_x = x;
      m_y = y;
      m_springX = springX;
      m_springY = springY;
   }//constructor
```

```
   /* constructor
*/
   public FailedAgentX(AgentX agent){
      m_agent = agent;
      m_id = agent.getID();
      m_x = agent.getX();
      m_y = agent.getY();
      m_springX = agent.getSpringX();
      m_springY = agent.getSpringY();

      m_oldDegree = agent.getDegree();
      agent.removeAllNeighbors();
   }//constructor

   /* This function returns the agent object for the agent this failed agent
*/
   /*   object is filling in for
*/
   public AgentX getDeadAgent(){
      return m_agent;
   }//getDeadAgent

   /* This function sets the value for the rate the agent adapts its
*/
   /*   connections to agents in the network
*/
   public void setAdaptationRate(int rate){
   }//setAdaptationRate

   /* This function writes the specified string to the parent TestAON log
*/
   public void writeLog(String line){
   }//writeLog

   /* This function writes the data for the agent in the comma seperated file
*/
   /*   that coincides with the specified system iteration and performance
*/
   public void writeAgentData(int iteration, float performance){
   }//writeAgentData

   /* This function saves the comma seperated file containing the agent's
*/
   /*   information for each time step to the specified file name
*/
   public void saveAgentData(String filename){
   }//saveAgentData

   /* This function returns the agent's id number
*/
   public int getID(){
      return m_id;
   }//getID

   /* This function returns the agent's id and skill id in the string format
*/
   /*   'id/skill_id'
*/
   public String getStringID(){
      return String.valueOf(m_id);
   }//getID
```

```
   /* This function returns the id of the task the agent is committed to
*/
   public int getTaskID(){
      return -1;
   }//getTaskID

   /* This function sets the id of the task the agent is committed to
*/
   public void setTaskID(int taskID){
   }//setTaskID

   /* This function returns the agent's x pixel location
*/
   public int getX(){
      return m_x;
   }//getX

   /* This function sets the agent's x pixel location
*/
   public void setX(int x){
      m_x = x;
   }//setX

   /* This function returns the agent's y pixel location
*/
   public int getY(){
      return m_y;
   }//getY

   /* This function sets the agent's y pixel location
*/
   public void setY(int y){
      m_y = y;
   }//setY

   /* This function returns the agent's spring adjusted x pixel location
*/
   public int getSpringX(){
      return m_springX;
   }//getSpringX

   /* This function returns the agent's spring adjusted y pixel location
*/
   public int getSpringY(){
      return m_springY;
   }//getSpringY

   /* This function returns the agent's type description string
*/
   public String getType(){
      return "Failed";
   }//getType

   /* This function returns the agent's state
*/
   public int getState(){
      return -1;
   }//getState

   /* This function returns the agent's work catagory
*/
   /*
*/
```

```
   public int getWorkCatagory(){
      return C_WAITED;
   }//getWorkCatagory

   /* This function returns the color to draw the agent's oval
*/
   public Color getColor(){
      return Color.black;
   }//getColor

   /* This function sets the maximum number of connections the agent can have
*/
   /*   to neighboring agents
*/
   public void setMaxDegree(int maxDegree){
   }//setMaxDegree

   /* This function returns whether the agent is available to commit to a
*/
   /*   task
*/
   public boolean isAvailable(){
      return false;
   }//isAvailable

   /* This function returns the color of the connection between the agent and
*/
   /*   another agent with the specified agent id
*/
   public Color getNeighborColor(int neighborID){
      Color color = Color.black;
      return color;
   }//getNeighborColor

   /* This function returns the number of neighbor connections the agent has
*/
   /*   changed by rewiring
*/
   public int getChangedEdgesCount(){
      return 0;
   }//getChangedEdgesCount

   /* This function returns a string listing the agent id and neighbor rating
*/
   /*   of each neighbor the agent has
*/
   public String getNeighborsString(){
      String result = "";
      return result;
   }//getNeighborsString

   /* This function returns a string representation of the agent's skill
*/
   public String getSkillsString(){
      return " ";
   }//getSkillsString

   /* This function returns the number of agents within the specified number
*/
   /*   of network connections of the agent
*/
   public int getDensity(int level){
      return 0;
```

```
   }//getDensity

   /* This function helps compute neighborhood density by adding neighbors
*/
   /*   to the agents touched list until the level variable reaches 0
*/
   public void getDensity(int level, Vector agentsTouched){
   }//getDensity

   /* This function returns the number of network connections the agent has
*/
   /*   to other agents in the network
*/
   public int getDegree(){
      return 0;
   }//getDegree

   /* This function returns the list of neighboring agents for this agent
*/
   public NeighborList getNeighbors(){
      return new NeighborList();
   }//getNeighbors

   /* This function returns true if the specified agent id identifies one of
*/
   /*   the agents this agent is connected to by network connections
*/
   public boolean hasNeighbor(int agentID){
      return false;
   }//hasNeighbor

   /* This function returns the skill the agent possesses
*/
   public int getSkills(){
      return 0;
   }//getSkills

   /* This function returns the string that identifies the team for the task
*/
   /*   the agent is committed to
*/
   public String getTeamString(){
      return m_teamString;
   }//getTeamString

   /* This function sets the flag that determines whether the agent changes
*/
   /*   its network connections to its neighbors
*/
   public void setAdaptNetwork(boolean adapt){
   }//setAdaptNetwork

   /* This function returns the agent with the specified id in the agent list
*/
   public AgentX getAgent(int id){
      return null;
   }//getAgent

   /* This function returns the number of agents in the agent list
*/
   public int getAgentCount(){
      return 0;
   }//getAgentCount
```

```
   /* This function returns the percentage of successful teams this agent has
*/
   /*   joined
*/
   public float getPerformance(){
      return 0.0f;
   }//getPerformance

   /* This function returns the average performance estimate for the agent's
*/
   /*   neighbors
*/
   public float getAverageNeighborPerformance(){
      return 0.0f;
   }//getAverageNeighborPerformance

   /* This function returns the percentage of the time the agent has been
*/
   /*   actively working on tasks
*/
   public float getPercentActive(){
      return 0.0f;
   }//getPercentActive

   /* This function returns the percentage of the time the agent has been
*/
   /*   waiting
*/
   public float getPercentWaiting(){
      return 0.0f;
   }//getPercentWaiting

   /* This function notifies the agent specified by the id that it has been
*/
   /*   added as a neighbor to this agent
*/
   public void sendAddNeighborMessage(int id){
   }//sendAddNeighborMessage

   /* This function adds the agent specified by the id as a neighbor to this
*/
   /*   agent
*/
   public boolean addNeighbor(int id){
      return false;
   }//addNeighbor

   /* This function notifies the agent specified by the id that it has been
*/
   /*   removed as a neighbor to this agent
*/
   public void sendRemoveNeighborMessage(int id){
   }//sendRemoveNeighborMessage

   /* This function removes the agent specified by the id as a neighbor of
*/
   /*   this agent
*/
   public void removeNeighbor(int id){
   }//removeNeighbor
```

```
   /* This function adds the agent specified by id as a neighbor to this
*/
   /*   agent without notifying the added neighbor
*/
   public void addNeighborNoNotify(int id){
   }//addNeighborNoNotify

   /* This function removes the agent specified by id as a neighbor to this
*/
   /*   agent without notifying the removed neighbor
*/
   protected void removeNeighborNoNotify(int id){
   }//removeNeighbor

   /* This function is used by other agents to tell this agent that it has
*/
   /*   been added as a neighbor
*/
   public void notifyAddNeighbor(int id){
   }//notifyAddNeighbor

   /* This function is used by other agents to tell this agent that it has
*/
   /*   been removed as a neighbor
*/
   public void notifyRemoveNeighbor(int id){
   }//notifyRemoveNeighbor

   /* This function pauses execution until the user presses enter at the
*/
   /*   console (used for debugging purposes)
*/
   public void pauseTest(String message){
      try{
         char input = ' ';
         byte temp[] = new byte[1];
         System.out.print("\n" + message + "\nPress <enter> to continue...");
         while(input != '\n'){
            System.in.read(temp);
            input = (char)temp[0];
         }//while
      }catch(IOException ex){
      }//catch
   }//pauseTest

   /* This function sets the skill possessed by the agent
*/
   public void addSkill(int skill){
   }//addSkill

   /* This function is called to update the agent for the current time step
*/
   /*   The list of system tasks is passed in as an AONTaskList
*/
   public void update(AONTaskList tasks){
   }//update

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
   }//updateUncommitted
```

```
   /* This function returns a random neighbor agent id as a neighbor to be
*/
   /*   removed
*/
   public int pickRemoveNeighbor(){
      return 0;
   }//pickRemoveNeighbor

   /* This function returns the distance from x1,y1 to x2,y2
*/
   public int getDistance(int x1, int y1, int x2, int y2){
      return 0;
   }//getDistance

   /* This function updates the agent when its state is committed
*/
   public void updateCommitted(AONTaskList tasks){
   }//updateCommitted

   /* This function updates the agent when its state is active
*/
   public void updateActive(){
   }//updateActive

   /* This function increments the life span of the agent's neighbor
*/
   /*   connections
*/
   public void incrementNeighborConnections(){
   }//incrementNeighborConnections

   /* This function returns the agent id of the most highly recommended
*/
   /*   agent who may become this agent's new neighbor
*/
   public int pickNewNeighbor(){
      int bestAgentID = -1;
      return bestAgentID;
   }//pickNewNeighbor

   /* This function returns whether the agent has chosen to join a team
*/
   /*   by committing to a task that another agent has proposed
*/
   public boolean joinTeam(AONTaskList tasks){
      return false;
   }//joinTeam

   /* This function returns true if the agent has accepted the specified
*/
   /*   task
*/
   public boolean acceptTask(AONTask task){
      return false;
   }//acceptTask

   /* This function returns the percentage of the agent's neighbors who are
*/
   /*    not committed to a task
*/
   public float percentNeighborsUncommitted(){
      return 0.0f;
   }//percentNeighborsUncommitted
```

```
   /* This function returns the index of the first unfilled skill in the
*/
   /*   specified task, or -1 if all skills are filled in the task
*/
   public int skillNeededInTask(AONTask task){
      return -1;
   }//skillNeededInTask

   /* This function returns the index of the first neighbor found to be
*/
   /*   committed to the specified task, or -1 if no neighbor is committed
*/
   /*   to the specified task
*/
   public int neighborCommitted(AONTask task){
      return -1;
   }//neighborCommitted

   /* This function returns the first task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickNeighborTask(AONTaskList tasks){
      return null;
   }//pickNeighborTask

   /* This function returns the task with the best percentage of skills
*/
   /*   that are filled (the closest to being actively worked on)
*/
   public AONTask pickBestNeighborTask(AONTaskList tasks){
      AONTask bestTask = null;
      return bestTask;
   }//pickBestNeighborTask

   /* This function returns an array containing the number of agents in the
*/
   /*   neighbor list that possess each skill
*/
   public int [] getNeighborTally(){
      int [] tally = new int [parent.getSkillMax()];
      return tally;
   }//getNeighborTally

   /* This function returns the first task which this agent is needed for
*/
   public AONTask pickTask(AONTaskList tasks){
      return null;
   }//pickTask

   /* This function returns the best task for this agent based on the skills
*/
   /*   already filled in the task and the skill to be filled by the agent
*/
   public AONTask pickBestTask(AONTaskList tasks){
      AONTask bestTask = null;
      return bestTask;
   }//pickBestTask

   /* This function returns the percentage of skills filled in the specified
*/
```

```
   /*   task by the skills available in the agents who are neighbors of this
*/
   /*   agent
*/
   public float skillMatchRatio(AONTask task){
      return 0.0f;
   }//skillMatchRatio

   /* This function updates the spring adjusted x and y pixel positions for
*/
   /*   this agent based on simulated spring forces on the connections to its
*/
   /*   neighbors
*/
   public void adjustLocation(int maxX, int maxY){
   }//adjustLocation

   /* This function returns the number of neighbors this agent has who are
*/
   /*   not committed to a task
*/
   public int getUnCommittedNeighborCount(){
      int count = 0;
      return count;
   }//getUnCommittedNeighborCount

   /* This function returns the number of neighbors this agent has who are
*/
   /*   committed to a task
*/
   public int getCommittedNeighborCount(){
      int count = 0;
      return count;
   }//getCommittedNeighborCount

   /* This function returns the number of neighbors this agent has who are
*/
   /*   actively working on the task they are committed to
*/
   public int getActiveNeighborCount(){
      return 0;
   }//getActiveNeighborCount

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      return false;
   }//chooseToDropTask

   /* This function returns a skill comparison weight between 0.0 and 1.0
*/
   /*   A value of 0.0 means their is no overlap in the skill sets
*/
   /*   A value of 1.0 means the skill sets are disjoint
*/
   public float getSkillComparison(int skill){
      return rand.nextFloat();
   }//getSkillComparison

   /* This function returns whether or not the agent has decided to adapt its
*/
```

```
   /*   network connections to its neighbors
*/
   public boolean chooseToAdapt(){
      return false;
   }//chooseToAdapt

   /* This function changes the agents network connections by getting a
*/
   /*   recommendation of a new neighbor from its current neighbors, then
*/
   /*   removing a connection to one of its current neighbors
*/
   public int rewire(){
      return -1;
   }//rewire

   /* This function resets the statistics kept about tasks neighbors have
*/
   /*   joined (Implemented by Performance agent)
*/
   public void resetTaskJoinCount(){
   }//resetTaskJoinCount

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String output = "";
      output = output + "FailedAgent " + m_id;
      return output;
   }//toString
}//class FailedFailedAgentX

----------------------------FailedAgentX.java-----------------------------------
-

----------------------------GnuplotFile.java-----------------------------------
-

import java.util.*;
import java.io.IOException;
import java.io.BufferedWriter;
import java.io.FileWriter;

/**
 * GnuplotFile Class
 *
 *
 */
public class GnuplotFile{
   String m_plotString;      /*
*/

   float m_xMin;             /*
*/
   float m_yMin;             /*
*/
   float m_xMax;             /*
*/
   float m_yMax;             /*
*/

   String m_title;           /*
*/
```

```
   String m_subtitle;        /*
*/
   String m_xLabel;          /*
*/
   String m_yLabel;          /*
*/

   int m_fontSize;           /*
*/

   /* constructor
*/
   public GnuplotFile(){
      m_plotString = "";

      m_xMin = 0.0f;
      m_yMin = 0.0f;
      m_xMax = 0.0f;
      m_yMax = 0.0f;
      m_title = "";
      m_subtitle = "";
      m_xLabel = "";
      m_yLabel = "";
      m_fontSize = 18;
   }//constructor

   public float get_xMin(){
      return m_xMin;
   }//get_xMin

   public void set_xMin(int xMin){
      m_xMin = xMin;
   }//set_xMin

   public float get_yMin(){
      return m_yMin;
   }//get_yMin

   public void set_yMin(int yMin){
      m_yMin = yMin;
   }//set_yMin

   public float get_xMax(){
      return m_xMax;
   }//get_xMax

   public void set_xMax(int xMax){
      m_xMax = xMax;
   }//set_xMax

   public float get_yMax(){
      return m_yMax;
   }//get_yMax

   public void set_yMax(int yMax){
      m_yMax = yMax;
   }//set_yMax

   public String getTitle(){
      return m_title;
   }//getTitle

   public void setTitle(String title){
```

```
      m_title = title;
   }//setTitle

   public String getSubtitle(){
      return m_subtitle;
   }//getSubtitle

   public void setSubtitle(String subtitle){
      m_subtitle = subtitle;
   }//setSubtitle

   public String getXLabel(){
      return m_xLabel;
   }//getXLabel

   public void setXLabel(String label){
      m_xLabel = label;
   }//setXLabel

   public String getYLabel(){
      return m_yLabel;
   }//getYLabel

   public void setYLabel(String label){
      m_yLabel = label;
   }//setYLabel

   public int getFontSize(){
      return m_fontSize;
   }//getFontSize

   public void setFontSize(int size){
      m_fontSize = size;
   }//setFontSize

   private int getMaxTimeStep(CSVFile file){
      int maxTimeStep = file.getLineCount()-1;
      try{
         String value = file.getItem(file.getLineCount()-1,0);
         int temp = (int)Float.parseFloat(value);
         maxTimeStep = temp;
      }catch(Exception ex){
      }//catch
      return maxTimeStep;
   }//getMaxTimeStep

   private void addPlotHeader(){
      m_plotString = m_plotString + "set title \"" + m_title + "\\n " +
m_subtitle + "\"\n";
      m_plotString = m_plotString + "set xrange[" + m_xMin + ":" + m_xMax +
"]\n";
      m_plotString = m_plotString + "set yrange[" + m_yMin + ":" + m_yMax +
"]\n";
      m_plotString = m_plotString + "set xlabel '" + m_xLabel + "'\n";
      m_plotString = m_plotString + "set ylabel '" + m_yLabel + "'\n";
      m_plotString = m_plotString + "set datafile separator \",\"\n";
      m_plotString = m_plotString + "set style line\n";
   }//addPlotHeader

   private void addPlotCommand(String directoryName, String outputName){
      m_plotString = m_plotString + "set terminal emf  'Times Roman Bold' " +
m_fontSize + "\n";
```

```
      m_plotString = m_plotString + "set output \"" + directoryName + "_" +
outputName + ".emf\"\n";
      m_plotString = m_plotString + "replot\n";
      m_plotString = m_plotString + "set output\n";
      m_plotString = m_plotString + "set terminal windows\n";
      m_plotString = m_plotString + "reset\n";
   }//addPlotCommand

   public void plotLinesPoints(CSVFile Results, Vector<String> testNames,
String inputName, String outputName, String directoryName){
      m_xMax = getMaxTimeStep(Results);
      m_yMax = Results.getMaxValue() + Results.getMaxValue()*0.3f;
      if(m_yMax == 0.0f){
         m_yMax = 0.1f;
      }//if

      addPlotHeader();
      for(int i = 0; i < getLength(testNames); i++){
         String plotString = "";
         String testName = (String)getItem(i,testNames);
         if(i == 0){
            plotString = "plot '" + inputName + directoryName + ".csv' using
1:" + (i+2) + " title \"" + testName + "\" with linespoints";
         }else{
            m_plotString = m_plotString + ",\\\n";
            plotString = "     '" + inputName + directoryName + ".csv' using
1:" + (i+2) + " title \"" + testName + "\" with linespoints";
         }//else
         m_plotString = m_plotString + plotString;
      }//for
      m_plotString = m_plotString + "\n";
      addPlotCommand(directoryName,outputName);
   }//plotLinesPoints

   public void plotYErrorBars(CSVFile Results, Vector<String> testNames, String
inputName, String outputName, String directoryName){
      m_xMax = getMaxTimeStep(Results);
      m_yMax = Results.getMaxValue() + Results.getMaxValue()*0.3f;
      if(m_yMax == 0.0f){
         m_yMax = 0.1f;
      }//if

      addPlotHeader();
      for(int i = 0; i < getLength(testNames); i++){
         String plotString = "";
         String testName = (String)getItem(i,testNames);
         if(i == 0){
            plotString = "plot '" + inputName + directoryName + ".csv' using
1:" + (i+2) + ":" + (i+2+getLength(testNames)) + " title \"" + testName + "\"
with yerrorbars";
         }else{
            m_plotString = m_plotString + ",\\\n";
            plotString = "     '" + inputName + directoryName + ".csv' using
1:" + (i+2) + ":" + (i+2+getLength(testNames)) + " title \"" + testName + "\"
with yerrorbars";
         }//else
         m_plotString = m_plotString + plotString;
      }//for
      m_plotString = m_plotString + "\n";
      addPlotCommand(directoryName,outputName);
   }//plotYErrorBars

   public void save(String filename){
```

```
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(filename));
         out.write(m_plotString);
         out.close();
         System.out.println("Wrote file: " + filename);
      }catch(IOException ex){
         System.out.println("Error writing Gnuplot file " + filename + ": " +
ex.getMessage());
      }//catch
   }//save

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getLength
}//GnuplotFile

----------------------------GnuplotFile.java-----------------------------------
-

---------------------------InventoryAgent.java---------------------------------
-

import java.util.*;

/**
 * InventoryAgent Class
 *   This is an agent that seeks to rewire to agents who have skills that
 *   have caused tasks to which the agent has been committed to fail or to
 *   agents who have skills that are not possessed by agents who are currently
 *   neighbors of this agent
 */
public class InventoryAgent extends AgentX{
```

```
   /* constructor
*/
   public InventoryAgent(TestAON p,int id, int x, int y, Vector agents){
      super(p,id,x,y,"Inventory",agents);
   }//constructor

   /* This function is used to find the best agent recommendation for a new
*/
   /*   neighbor by doing a recursive search of the network until depth has
*/
   /*   reached 0. The best agent recommendation is returned back to the
*/
   /*   agent who originated the request for agent recommendations
*/
   public AgentRecommendation recommendNeighbor(int depth, Vector
agentsVisited, AgentX sourceAgent){
      int bestAgentID = -1;
      float bestCompareValue = -1.0f;
      depth--;
      agentsVisited.add(m_id);
      if(depth <= 0){
         if(!sourceAgent.hasNeighbor(m_id) && m_id != sourceAgent.getID()){
            bestAgentID = m_id;
            bestCompareValue = sourceAgent.getSkillComparison(getSkills());
         }//if
      }else{
         for(int i = 0; i < m_neighbors.getLength(); i++){
            int id = m_neighbors.getItem(i).agentID();
            if(!contains(id,agentsVisited)){
               AgentX neighbor = (AgentX)getAgent(id);
               AgentRecommendation agentRecommendation =
neighbor.recommendNeighbor(depth,agentsVisited,sourceAgent);
               if(agentRecommendation.getID() >= 0){
                  int agentDegree =
getAgent(agentRecommendation.getID()).getDegree();
                  if((float)agentRecommendation.getValue() > bestCompareValue
&& agentDegree < m_maxDegree){
                     bestAgentID = agentRecommendation.getID();
                     bestCompareValue = agentRecommendation.getValue();
                  }//if
               }//if
            }//if
         }//for
         if(!sourceAgent.hasNeighbor(m_id) && m_id != sourceAgent.getID()){
            if(bestAgentID == -1){
               bestAgentID = m_id;
               bestCompareValue = sourceAgent.getSkillComparison(getSkills());
            }else{
               float personalComparison =
sourceAgent.getSkillComparison(getSkills());
               if(personalComparison > bestCompareValue){
                  bestAgentID = m_id;
                  bestCompareValue = personalComparison;
               }//if
            }//else
         }//if
      }//else
      return new AgentRecommendation(bestAgentID,bestCompareValue);
   }//recommendNeighbor

   /* This function returns a skill comparison weight between 0.0 and 1.0
*/
```

```
   /*   A value of 0.0 means their have been no tasks fail with the
*/
   /*   specified skill unfilled.
*/
   /*   Larger values mean there have been more tasks fail with the specified
*/
   /*   skill unfilled
*/
   /*   If no agents have the specified skill, then the comparison is given a
*/
   /*   very large value
*/
   public float getSkillComparison(int skill){
      int unfilledSkillCount = 0;
      int neighborhoodSkillCount = 1;

      if(m_skillHistoryLength <= 0){
         if(m_unfilledSkillsCount != null){
            unfilledSkillCount = (Integer)getItem(skill,m_unfilledSkillsCount);
         }//if
      }else{
         if(m_unfilledSkillHistory != null){
            unfilledSkillCount =
m_unfilledSkillHistory.getSkillShortage(skill);
         }//if
      }//else

      if(m_neighborhoodStats != null){
         neighborhoodSkillCount =
(Integer)getItem(skill,m_neighborhoodStats.getSkillCount());
      }//if

      if(neighborhoodSkillCount == 0){
         return 10000.0f;
      }//if
      return (float)unfilledSkillCount/neighborhoodSkillCount;
   }//getSkillComparison

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "InventoryAgent";
      if(m_neighborhoodStats != null){
         value = value + "\n" + m_neighborhoodStats.toString();
      }//if
      value = value + "\n" + super.toString() + "\n";
      return value;
   }//toString
}//class InventoryAgent

--------------------------InventoryAgent.java-----------------------------------
-

----------------------InventoryImpatientAgent.java-----------------------------
-

import java.util.*;

/**
 * InventoryImpatientAgent Class
 *   This is an agent that seeks to rewire to agents who have skills that
 *   have caused tasks to which the agent has been committed to fail or to
 *   agents who have skills that are not possessed by agents who are currently
```

```
 *   neighbors of this agent
 *   -This agent is 'impatient' because it builds off the Inventory agent
 *     by choosing to drop its commitment to a task when the skills left to be
 *     filled in the task exceed the number of neighboring agents who are not
 *     committed to a task
 */
public class InventoryImpatientAgent extends InventoryAgent{
   /* constructor
*/
   public InventoryImpatientAgent(TestAON p,int id, int x, int y, Vector
agents){
      super(p,id,x,y,agents);
      m_type = "InventoryImpatient";
   }//constructor

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      boolean dropTask = false;
      if(getUnCommittedNeighborCount() < task.unFilledSkillsCount()){
         dropTask = true;
      }//if
      return dropTask;
   }//chooseToDropTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "InventoryImpatientAgent\n" + super.toString() + "\n";
      return value;
   }//toString
}//class InventoryImpatientAgent

----------------------InventoryImpatientAgent.java-----------------------------
-

------------------------InventoryStratAgent.java-------------------------------
-

import java.util.*;

/**
 * InventoryStratAgent Class
 *   This is an agent that seeks to rewire to agents who have skills that
 *   have caused tasks to which the agent has been committed to fail or to
 *   agents who have skills that are not possessed by agents who are currently
 *   neighbors of this agent
 *   -This agent is 'strategic' because it builds off the Inventory agent
 *    by choosing tasks to join and tasks to propose that are the best match
 *    for the skills possessed by the agent's uncommitted neighbors
 */
public class InventoryStratAgent extends InventoryAgent{
   /* constructor
*/
   public InventoryStratAgent(TestAON p,int id, int x, int y, Vector agents){
      super(p,id,x,y,agents);
      m_type = "InventoryStrat";
   }//constructor

   /* This function updates the agent when its state is uncommitted
*/
```

```
   public void updateUncommitted(AONTaskList tasks){
       m_edgesChanged = 0;
       writeLog(m_id + " updateUncommitted:");
       // Look at all tasks which your immediate neighbor has committed to
       // Select the one which is best based on (1) one with more committed
agents
       // (2) one for which your neighbors have the most chance of satisfying
       AONTask taskJoin = pickNeighborTask(tasks);
       if(taskJoin != null){
          acceptTask(taskJoin);
          m_state = C_COMMITTED;
          m_numberOfTeamsJoined++;
          m_committedTimeRemaining = m_maxCommittedTime;
          m_lastWorkCatagory = C_JOINED;
         return;
      }//if

      float random = rand.nextFloat();
      // WaitPercent can be fixed or dynamic (based on success with current
strategy)
      Boolean wantToWait = (random < m_waitPercent);
      if(chooseToAdapt()){
         rewire();
      }else if(wantToWait){
         return;
      }else{
         //Propose a new task
         // Look at all available tasks which you can do.
         // Select the one which is best based on (1) random choice
         // (2) one which your neighbors have the most chance of satisfying
         AONTask task = pickTask(tasks);
         if(task != null){
            acceptTask(task);
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
            m_lastWorkCatagory = C_PROPOSED;
            m_numberOfTeamsJoined++;
         }//if
      }//else
   }//updateUncommitted

   /* This function returns the best task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickNeighborTask(AONTaskList tasks){
      return pickBestNeighborTask(tasks);
   }//pickNeighborTask

   /* This function returns the best task which this agent is needed for
*/
   public AONTask pickTask(AONTaskList tasks){
      return pickBestTask(tasks);
   }//pickTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "InventoryStratAgent\n" + super.toString() + "\n";
      return value;
   }//toString
}//class InventoryStratAgent
```

```
------------------------InventoryStratAgent.java-------------------------------
-

--------------------InventoryStratImpatientAgent.java--------------------------
-

import java.util.*;

/**
 * InventoryStratImpatientAgent Class
 *   This is an agent that seeks to rewire to agents who have skills that
 *   have caused tasks to which the agent has been committed to fail or to
 *   agents who have skills that are not possessed by agents who are currently
 *   neighbors of this agent
 *   -This agent is 'strategic' because it builds off the Inventory agent
 *    by choosing tasks to join and tasks to propose that are the best match
 *    for the skills possessed by the agent's uncommitted neighbors
 *   -This agent is 'impatient' because it builds off the Inventory agent
 *     by choosing to drop its commitment to a task when the skills left to be
 *     filled in the task exceed the number of neighboring agents who are not
 *     committed to a task
 */
public class InventoryStratImpatientAgent extends InventoryAgent{
   /* constructor
*/
   public InventoryStratImpatientAgent(TestAON p,int id, int x, int y, Vector
agents){
      super(p,id,x,y,agents);
      m_type = "InventoryStratImpatient";
   }//constructor

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
       m_edgesChanged = 0;
       writeLog(m_id + " updateUncommitted:");
       // Look at all tasks which your immediate neighbor has committed to
       // Select the one which is best based on (1) one with more committed
agents
       // (2) one for which your neighbors have the most chance of satisfying
       AONTask taskJoin = pickNeighborTask(tasks);
       if(taskJoin != null){
          acceptTask(taskJoin);
          m_state = C_COMMITTED;
          m_numberOfTeamsJoined++;
          m_committedTimeRemaining = m_maxCommittedTime;
          m_lastWorkCatagory = C_JOINED;
         return;
      }//if

      float random = rand.nextFloat();
      // WaitPercent can be fixed or dynamic (based on success with current
strategy)
      Boolean wantToWait = (random < m_waitPercent);
      if(chooseToAdapt()){
         rewire();
      }else if(wantToWait){
         return;
      }else{
         //Propose a new task
         // Look at all available tasks which you can do.
         // Select the one which is best based on (1) random choice
         // (2) one which your neighbors have the most chance of satisfying
```

```
         AONTask task = pickTask(tasks);
         if(task != null){
            acceptTask(task);
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
            m_lastWorkCatagory = C_PROPOSED;
            m_numberOfTeamsJoined++;
         }//if
      }//else
   }//updateUncommitted

   /* This function returns the best task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickNeighborTask(AONTaskList tasks){
      return pickBestNeighborTask(tasks);
   }//pickNeighborTask

   /* This function returns the best task which this agent is needed for
*/
   public AONTask pickTask(AONTaskList tasks){
      return pickBestTask(tasks);
   }//pickTask

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      boolean dropTask = false;
      if(getUnCommittedNeighborCount() < task.unFilledSkillsCount()){
         dropTask = true;
      }//if
      return dropTask;
   }//chooseToDropTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "InventoryStratImpatientAgent\n" + super.toString() +
"\n";
      return value;
   }//toString
}//class InventoryStratImpatientAgent

--------------------InventoryStratImpatientAgent.java--------------------------
-

--------------------------MyMessageFrame.java---------------------------------
-

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

/**
 * MyMessageFrame Class
 *
 *
 */
public class MyMessageFrame extends JFrame implements Runnable{
```

```
   private TextArea messageText = new TextArea();
   private Frame parentFrame;/*
*/

   private static int keyEventValue = 0;
   private boolean running;  /*
*/

   private int lines;        /*
*/
   private int MAX_LINES;    /*
*/

   private String filename;  /*
*/
   private boolean append;   /*
*/

   /* constructor
*/
   public MyMessageFrame(String title, int width, int height, int key, Frame
parent){
      super(title);
      setSize(width,height);//200,100);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            setVisible(false);
            //System.exit(0);
         }//windowClosing
      });

      addKeyListener(new KeyAdapter(){
         public void keyPressed(KeyEvent ke){
            if(ke.getKeyCode() == keyEventValue){
               setVisible(false);
            }//if
         }//keyPressed
      });//KeyListener

      parentFrame = parent;
      keyEventValue = key;
      running = false;

      messageText.setEditable(false);
      this.getContentPane().add(messageText);

      lines = 0;
      MAX_LINES = 300;

      filename = "DefaultMessageFrame_log.txt";
      append = false;
   }//constructor

   public void run(){
      running = true;
   }//run

   public boolean isRunning(){
      return running;
   }//isRunning

   public boolean isVisible(){
      return super.isVisible();
```

```
   }//isVisible

   public void setVisible(boolean v){
      super.setVisible(v);
      requestFocus();
   }//setVisible

   public void addText(String text){
      messageText.append(text+"\n");
      if(parentFrame != null){
         parentFrame.requestFocus();
      }//if
      if(lines >= MAX_LINES){
         saveContentsToFile(filename);
         append = true;
         clearText();
         lines = 0;
      }//if
   }//addText

   public void clearText(){
      messageText.setText(null);
   }//clearText

   public void setFilename(String fname){
      filename = fname;
   }//setFilename

   public void saveContentsToFile(String fname){
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(fname,append));
         out.write(messageText.getText());
         out.close();
         System.out.println("Wrote contents to " + fname);
      }catch(IOException e){
         System.out.println("IOException while writing message frame
contents.");
      }//catch
   }//saveContentsToFile

   public void saveContentsToFile(String fname, boolean exitAfterWrite){
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(fname,append));
         out.write(messageText.getText());
         out.close();
         System.out.println("Wrote contents to " + fname);
      }catch(IOException e){
         System.out.println("IOException while writing message frame
contents.");
      }//catch
      if(exitAfterWrite){
         System.exit(0);
      }//if
   }//saveContentsToFile
}//class MyMessageFrame

--------------------------MyMessageFrame.java---------------------------------
-

-------------------------NeighborhoodStats.java-------------------------------
-

import java.util.*;
```

```
/**
 * NeighborhoodStats Class
 *
 *
 */
public class NeighborhoodStats{
   int m_agentCount;         /*
*/
   float m_sumPerformance;   /*
*/
   Vector<Integer> m_skillCount;
   Vector<Integer> m_skillDistance;

   /* constructor
*/
   public NeighborhoodStats(int systemSkillCount){
      m_agentCount = 0;
      m_sumPerformance = 0.0f;
      m_skillCount = new Vector();
      m_skillDistance = new Vector();
      for(int i = 0; i < systemSkillCount; i++){
         m_skillCount.add(0);
         m_skillDistance.add(-1);
      }//for
   }//constructor

   public void addNeighborStats(int skillID, int skillDistance, float
performance){
      m_agentCount++;
      m_sumPerformance += performance;
      int count = m_skillCount.elementAt(skillID);
      m_skillCount.setElementAt(count+1,skillID);
      int distance = m_skillDistance.elementAt(skillID);
      if(distance < 0){
         m_skillDistance.setElementAt(skillDistance,skillID);
      }//if
   }//addSkillCount

   public float averagePerformance(){
      return m_sumPerformance/m_agentCount;
   }//averagePerformance

   public Vector<Integer> getSkillCount(){
      return m_skillCount;
   }//getSkillCount

   public Vector<Integer> getSkillDistance(){
      return m_skillDistance;
   }//getSkillDistance

   public String toString(){
      String value = "NeighborhoodStats: ";
      int skillCount = m_skillCount.size();
      for(int i = 0; i < skillCount; i++){
         value = value + "\n  skill " + i + ": count=" +
m_skillCount.elementAt(i);
         value = value + " distance=" + m_skillDistance.elementAt(i);
      }//for
      return value;
   }//toString
}//NeighborhoodStats
```

```
-------------------------NeighborhoodStats.java--------------------------------
-

---------------------------NeighborLink.java-----------------------------------
-

/**
 * NeighborLink Class
 *
 *
 */
public class NeighborLink{
   int m_agentID;            /*
*/
   float m_agentRating;      /*
*/
   int m_connectionLife;     /*
*/

   /* constructor
*/
   public NeighborLink(){
      m_agentID = -1;
      m_agentRating = 0.0f;
      m_connectionLife = 0;
   }//constructor

   /* constructor
*/
   public NeighborLink(int agentID){
      m_agentID = agentID;
      m_agentRating = 0.0f;
      m_connectionLife = 0;
   }//constructor

   public int agentID(){
      return m_agentID;
   }//agentID

   public float agentRating(){
      return m_agentRating;
   }//agentRating

   public int connectionLife(){
      return m_connectionLife;
   }//connectionLife

   public void setAgentRating(float rating){
      m_agentRating = rating;
   }//setAgentRating

   public void setConnectionLife(int life){
      m_connectionLife = life;
   }//setConnectionLife

   public String toString(){
      return "[NeighborLink id=" + m_agentID + " rating=" + m_agentRating + "
connectionLife=" + m_connectionLife + "]";
   }//toString
}//NeighborLink

-------------------------NeighborLink.java-----------------------------------
-
```

```
----------------------------NeighborList.java-----------------------------------
-

import java.util.Vector;

/**
 * NeighborhoodList Class
 *
 *
 */
public class NeighborList{
   Vector<NeighborLink> m_neighbors;

   /* constructor
*/
   public NeighborList(){
      m_neighbors = new Vector();
   }//constructor

   public void addAgent(int agentID){
      m_neighbors.add(new NeighborLink(agentID));
   }//addAgent

   public NeighborLink getItem(int index){
      return m_neighbors.elementAt(index);
   }//getItem

   public void setItem(int index, NeighborLink item){
      m_neighbors.setElementAt(item,index);
   }//setItem

   public boolean containsAgent(int agentID){
      for(int i = 0; i < m_neighbors.size(); i++){
         NeighborLink neighbor = m_neighbors.elementAt(i);
         if(neighbor.agentID() == agentID){
            return true;
         }//if
      }//for
      return false;
   }//containsAgent

   public NeighborLink getNeighbor(int agentID){
      for(int i = 0; i < m_neighbors.size(); i++){
         NeighborLink neighbor = m_neighbors.elementAt(i);
         if(neighbor.agentID() == agentID){
            return neighbor;
         }//if
      }//for
      return null;
   }//getNeighbor

   public void removeAgent(int agentID){
      for(int i = 0; i < m_neighbors.size(); i++){
         NeighborLink neighbor = m_neighbors.elementAt(i);
         if(neighbor.agentID() == agentID){
            m_neighbors.removeElementAt(i);
            return;
         }//if
      }//for
   }//removeAgent

   public int getLength(){
```

```
      return m_neighbors.size();
   }//getLength

   public void setRating(int index, float rating){
      NeighborLink neighbor = m_neighbors.elementAt(index);
      neighbor.setAgentRating(rating);
   }//setRating

   public void setAgentRating(int agentID, float rating){
      for(int i = 0; i < m_neighbors.size(); i++){
         NeighborLink neighbor = m_neighbors.elementAt(i);
         if(neighbor.agentID() == agentID){
            neighbor.setAgentRating(rating);
            return;
         }//if
      }//for
   }//setAgentRating

   public void setConnectionLife(int index, int life){
      NeighborLink neighbor = m_neighbors.elementAt(index);
      neighbor.setConnectionLife(life);
   }//setConnectionLife

   public int indexOfAgent(int agentID){
      for(int i = 0; i < m_neighbors.size(); i++){
         NeighborLink neighbor = m_neighbors.elementAt(i);
         if(neighbor.agentID() == agentID){
            return i;
         }//if
      }//for
      return -1;
   }//indexOfAgent

   public String toString(){
      String result = "";
      for(int i = 0; i < m_neighbors.size(); i++){
         NeighborLink neighbor = m_neighbors.elementAt(i);
         if(i == 0){
            result = neighbor.toString();
         }else{
            result = result + ", " + neighbor.toString();
         }//else
      }//for
      return result;
   }//toString
}//NeighborList

----------------------------NeighborList.java-----------------------------------
-

----------------------------NodeFailure.java------------------------------------
-

/**
 * NodeFailure Class
 *
 *
 */
public class NodeFailure{
   int m_agentID;            /*
*/
   int m_timeStep;           /*
*/
```

```
   int m_regenerationTime;   /*
*/

   /* constructor
*/
   public NodeFailure(){
      m_agentID = 0;
      m_timeStep = 0;
      m_regenerationTime = -1;
   }//constructor

   /* constructor
*/
   public NodeFailure(int agentID, int timeStep, int regenerationTime){
      m_agentID = agentID;
      m_timeStep = timeStep;
      m_regenerationTime = regenerationTime;
   }//constructor

   public int getAgentID(){
      return m_agentID;
   }//getAgentID

   public int getTimeStep(){
      return m_timeStep;
   }//getTimeStep

   public int getRegenerationTime(){
      return m_regenerationTime;
   }//getRegenerationTime

   public void decrementRegenerationTime(){
      m_regenerationTime--;
   }//decrementRegenerationTime

   public String toString(){
      return "AgentID=" + m_agentID + " TimeStep=" + m_timeStep + "
RegenerationTime=" + m_regenerationTime;
   }//toString
}//class NodeFailure

----------------------------NodeFailure.java-----------------------------------
-
```

```
-------------------------PerformanceAgent.java---------------------------------
-

import java.util.*;
import java.awt.*;

/**
 * PerformanceAgent Class
 *   This is an agent that seeks to rewire to agents who have the best
 *   estimated performance.
 */
public class PerformanceAgent extends AgentX{
   Vector<Integer> m_neighborsSuccess;
```

```
                              /* A count for the number of successful teams
*/
                              /* for each of the agent's neighbors
*/
   Vector<Integer> m_neighborsTotal;
                              /* A count for the number of total teams for
*/
                              /* each of the agent's neighbors
*/
   Hashtable m_teamsJoined;  /* A lookup table of task ids for teams the
*/
                              /* agent has joined in the past
*/
   int m_validJoinCount;     /* The minimum number of teams the agent must
*/
                              /* join before comparing its own performance
*/
                              /* with neighbor's performance when rewiring
*/

   /* constructor
*/
   public PerformanceAgent(TestAON p,int id, int x, int y, Vector agents){
      super(p,id,x,y,"Performance",agents);

      m_adaptationRate = 5;

      m_neighborsSuccess = new Vector();
      m_neighborsTotal = new Vector();
      m_teamsJoined = new Hashtable();
      m_validJoinCount = 10;
      initializeTaskJoinCount();
   }//constructor

   /* This function returns whether or not the agent has decided to adapt its
*/
   /*   network connections to its neighbors
*/
   public boolean chooseToAdapt(){
      boolean adapt = false;
      float num = (float)m_adaptationRate/getAgentCount();
      float random = rand.nextFloat();
      if(random < num && m_adaptNetwork){
         adapt = true;
      }//if
      return adapt;
   }//chooseToAdapt

   /* This function adds the agent specified by the id as a neighbor to this
*/
   /*   agent
*/
   public boolean addNeighbor(int id){
       boolean res = super.addNeighbor(id);
       if(res){
          m_neighborsSuccess.add(new Integer(0));
          m_neighborsTotal.add(new Integer(0));
       }//if
       return res;
   }//addNeighbor

   /* This function removes the agent specified by the id as a neighbor of
*/
```

```
   /*   this agent
*/
   public void removeNeighbor(int id){
      if(m_neighbors.containsAgent(id)){
         m_neighbors.removeAgent(id);
         m_neighborsSuccess.remove(new Integer(id));
         m_neighborsTotal.remove(new Integer(id));
         sendRemoveNeighborMessage(id);
      }//if
   }//removeNeighbor

   /* This function adds the agent specified by id as a neighbor to this
*/
   /*   agent without notifying the added neighbor
*/
   public void addNeighborNoNotify(int id){
      if(!m_neighbors.containsAgent(id) && id != m_id){
         m_neighbors.addAgent(id);
         m_neighborsSuccess.add(new Integer(0));
         m_neighborsTotal.add(new Integer(0));
      }//if
   }//addNeighborNoNotify

   /* This function removes the agent specified by id as a neighbor to this
*/
   /*   agent without notifying the removed neighbor
*/
   protected void removeNeighborNoNotify(int id){
      if(m_neighbors.containsAgent(id)){
         m_neighbors.removeAgent(id);
         m_neighborsSuccess.remove(new Integer(id));
         m_neighborsTotal.remove(new Integer(id));
      }//if
   }//removeNeighborNoNotify

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
      float localPerformance = getPerformance(), averageNeighborPerformance =
getAverageNeighborPerformance();
      m_edgesChanged = 0;

      if(chooseToAdapt() && localPerformance < averageNeighborPerformance &&
m_numberOfTeamsJoined > m_validJoinCount && m_adaptNetwork){
         rewire();
      }else{
         if(joinTeam(tasks)){
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
         }//if
      }//else
   }//updateUncommitted

   /* This function updates the agent when its state is active
*/
   public void updateActive(){
      if(m_activeTimeRemaining > 0){
         m_activeTimeRemaining--;
      }//if
      if(m_activeTimeRemaining <= 0){
         countSuccessfulTask(m_taskID);
         m_state = C_UNCOMMITTED;
         m_taskID=-1;
```

```
      }else{
         m_lastWorkCatagory = C_WORKED;
      }//else
   }//updateActive

   /* This function is used to find the best agent recommendation for a new
*/
   /*   neighbor by doing a recursive search of the network until depth has
*/
   /*   reached 0. The best agent recommendation is returned back to the
*/
   /*   agent who originated the request for agent recommendations
*/
   public AgentRecommendation recommendNeighbor(int depth, Vector
agentsVisited, AgentX sourceAgent){
      int bestAgentID = -1;
      float bestPerformance = -1.0f;

      depth--;
      agentsVisited.add(m_id);
      if(depth <= 0){
         if(!sourceAgent.hasNeighbor(m_id) && m_id != sourceAgent.getID()){
            bestAgentID = m_id;
            if(m_numberOfTeamsJoined > 0){
               bestPerformance = m_numberOfSuccessfulTeamsJoined /
m_numberOfTeamsJoined;
            }else{
               bestPerformance = 0.0f;
            }//else
         }//if
      }else{
         for(int i = 0; i < m_neighbors.getLength(); i++){
            int id = m_neighbors.getItem(i).agentID();
            if(!contains(id,agentsVisited)){
               AgentX neighbor = (AgentX)getAgent(id);
               AgentRecommendation agentRecommendation =
neighbor.recommendNeighbor(depth,agentsVisited,sourceAgent);
               if(agentRecommendation.getID() >= 0){
                  int agentDegree =
getAgent(agentRecommendation.getID()).getDegree();
                  if(agentRecommendation.getValue() > bestPerformance &&
agentDegree < m_maxDegree){
                     bestAgentID = agentRecommendation.getID();
                     bestPerformance = agentRecommendation.getValue();
                  }//if
               }//if
            }//if
         }//for
         if(!sourceAgent.hasNeighbor(m_id) && m_id != sourceAgent.getID()){
            if(bestAgentID == -1){
               bestAgentID = m_id;
               if(m_numberOfTeamsJoined > 0){
                  bestPerformance = m_numberOfSuccessfulTeamsJoined /
m_numberOfTeamsJoined;
               }else{
                  bestPerformance = 0.0f;
               }//else
            }else{
               float personalPerformance = 0.0f;
               if(m_numberOfTeamsJoined > 0){
                  personalPerformance = m_numberOfSuccessfulTeamsJoined /
m_numberOfTeamsJoined;
               }//if
```

```
               if(personalPerformance > bestPerformance){
                  bestAgentID = m_id;
                  bestPerformance = personalPerformance;
               }//if
            }//else
         }//if
      }//else
      return new AgentRecommendation(bestAgentID,bestPerformance);
   }//recommendNeighbor

   /* This function counts the successful completion of a task by a team
*/
   public void countSuccessfulTask(int taskID){
      try{
         m_numberOfSuccessfulTeamsJoined++;
         if(m_teamsJoined.contains(new Integer(taskID))){
            int neighborID = ((Integer)m_teamsJoined.get(new
Integer(taskID))).intValue();
            int previous =
((Integer)getItem(neighborID,m_neighborsSuccess)).intValue();
            setItem(neighborID,m_neighborsSuccess,new Integer(previous+1));
         }//if
      }catch(Exception ex){
         System.out.println("Error counting successful task: " +
ex.toString());
      }//catch
   }//countSuccessfulTask

   /* This function initializes the nieghbor task total and task success
*/
   /*   vectors
*/
   public void initializeTaskJoinCount(){
      for(int i = 0; i < m_neighbors.getLength(); i++){
         m_neighborsSuccess.add(new Integer(0));
         m_neighborsTotal.add(new Integer(0));
      }//for
   }//initializeTaskJoinCount

   /* This function resets the statistics kept about tasks neighbors have
*/
   /*   joined
*/
   public void resetTaskJoinCount(){
      for(int i = 0; i < m_neighbors.getLength(); i++){
         m_neighborsSuccess.add(new Integer(0));
         m_neighborsTotal.add(new Integer(0));
      }//for
   }//resetTaskJoinCount

   /* This function returns a neighbor agent id for the worst performing
*/
   /*   neighbor as the neighbor to be removed
*/
   public int pickRemoveNeighbor(){
      float worsePerf = 100.0f;
      int worseAgent = -1;
      if(m_neighbors.getLength() == 0){
         return -1;
      }//if
      // Find the worst performance NOT counting the neighbor you just added
      for(int i = 0; i < m_neighbors.getLength()-1; i++){
         int id = m_neighbors.getItem(i).agentID();
```

```
         AgentX agent = (AgentX)getAgent(id);
         float perf = agent.getPerformance();
         if(perf < worsePerf){
            worsePerf = perf;
            worseAgent = id;
         }//if
      }//for

      return worseAgent;
   }//pickRemoveNeighbor

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "Performance_" + super.toString() + "\n";
      return value;
   }//toString
}//class PerformanceAgentAdv

-------------------------PerformanceAgent.java--------------------------------
-



--------------------PerformanceImpatientAgent.java----------------------------
-

import java.util.*;

/**
 * PerformanceImpatientAgent Class
 *   This is an agent that seeks to rewire to agents who have the best
 *   estimated performance.
 *   -This agent is 'impatient' because it builds off the Performance agent
 *     by choosing to drop its commitment to a task when the skills left to be
 *     filled in the task exceed the number of neighboring agents who are not
 *     committed to a task
 */
public class PerformanceImpatientAgent extends PerformanceAgent{
   /* constructor
*/
   public PerformanceImpatientAgent(TestAON p,int id, int x, int y, Vector
agents){
      super(p,id,x,y,agents);
      m_type = "PerformanceImpatient";
   }//constructor

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      boolean dropTask = false;
      if(getUnCommittedNeighborCount() < task.unFilledSkillsCount()){
         dropTask = true;
      }//if
      return dropTask;
   }//chooseToDropTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "PerformanceImpatient_" + super.toString() + "\n";
```

```
      return value;
   }//toString
}//class PerformanceImpatientAgent

--------------------PerformanceImpatientAgent.java-----------------------------
-

----------------------PerformanceStratAgent.java-----------------------------
-

import java.util.*;

/**
 * PerformanceStratAgent Class
 *   This is an agent that seeks to rewire to agents who have the best
 *   estimated performance.
 *   -This agent is 'strategic' because it builds off the Performance agent
 *    by choosing tasks to join and tasks to propose that are the best match
 *    for the skills possessed by the agent's uncommitted neighbors
 */
public class PerformanceStratAgent extends PerformanceAgent{
   /* constructor
*/
   public PerformanceStratAgent(TestAON p,int id, int x, int y, Vector agents){
      super(p,id,x,y,agents);
      m_type = "PerformanceStrat";
   }//constructor

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
       m_edgesChanged = 0;
       writeLog(m_id + " updateUncommitted:");
       // Look at all tasks which your immediate neighbor has committed to
       // Select the one which is best based on (1) one with more committed
agents
       // (2) one for which your neighbors have the most chance of satisfying
       AONTask taskJoin = pickNeighborTask(tasks);
       if(taskJoin != null){
          acceptTask(taskJoin);
          m_state = C_COMMITTED;
          m_numberOfTeamsJoined++;
          m_committedTimeRemaining = m_maxCommittedTime;
          m_lastWorkCatagory = C_JOINED;
         return;
      }//if

      float random = rand.nextFloat();
      // WaitPercent can be fixed or dynamic (based on success with current
strategy)
      Boolean wantToWait = (random < m_waitPercent);
      if(chooseToAdapt()){
         rewire();
      }else if(wantToWait){
         return;
      }else{
         //Propose a new task
         // Look at all available tasks which you can do.
         // Select the one which is best based on (1) random choice
         // (2) one which your neighbors have the most chance of satisfying
         AONTask task = pickTask(tasks);
         if(task != null){
            acceptTask(task);
```

```
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
            m_lastWorkCatagory = C_PROPOSED;
            m_numberOfTeamsJoined++;
         }//if
      }//else
   }//updateUncommitted

   /* This function returns the best task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickNeighborTask(AONTaskList tasks){
      return pickBestNeighborTask(tasks);
   }//pickNeighborTask

   /* This function returns the best task which this agent is needed for
*/
   public AONTask pickTask(AONTaskList tasks){
      return pickBestTask(tasks);
   }//pickTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "PerformanceStrat_" + super.toString() + "\n";
      return value;
   }//toString
}//class PerformanceStratAgent
```

----------PerformanceStratAgent.java----------
-

----------PerformanceStratImpatientAgent.java----------
-

```
import java.util.*;

/**
 * PerformanceStratAgent Class
 *   This is an agent that seeks to rewire to agents who have the best
 *   estimated performance.
 *   -This agent is 'strategic' because it builds off the Performance agent
 *    by choosing tasks to join and tasks to propose that are the best match
 *    for the skills possessed by the agent's uncommitted neighbors
 *   -This agent is 'impatient' because it builds off the Performance agent
 *     by choosing to drop its commitment to a task when the skills left to be
 *     filled in the task exceed the number of neighboring agents who are not
 *     committed to a task
 */
public class PerformanceStratImpatientAgent extends PerformanceAgent{
   /* constructor
*/
   public PerformanceStratImpatientAgent(TestAON p,int id, int x, int y, Vector
agents){
```

```
      super(p,id,x,y,agents);
      m_type = "PerformanceStratImpatient";
   }//constructor

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
       m_edgesChanged = 0;
       writeLog(m_id + " updateUncommitted:");
       // Look at all tasks which your immediate neighbor has committed to
       // Select the one which is best based on (1) one with more committed
agents
       // (2) one for which your neighbors have the most chance of satisfying
       AONTask taskJoin = pickNeighborTask(tasks);
       if(taskJoin != null){
          acceptTask(taskJoin);
          m_state = C_COMMITTED;
          m_numberOfTeamsJoined++;
          m_committedTimeRemaining = m_maxCommittedTime;
          m_lastWorkCatagory = C_JOINED;
         return;
      }//if

      float random = rand.nextFloat();
      // WaitPercent can be fixed or dynamic (based on success with current
strategy)
      Boolean wantToWait = (random < m_waitPercent);
      if(chooseToAdapt()){
         rewire();
      }else if(wantToWait){
         return;
      }else{
         //Propose a new task
         // Look at all available tasks which you can do.
         // Select the one which is best based on (1) random choice
         // (2) one which your neighbors have the most chance of satisfying
         AONTask task = pickTask(tasks);
         if(task != null){
            acceptTask(task);
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
            m_lastWorkCatagory = C_PROPOSED;
            m_numberOfTeamsJoined++;
         }//if
      }//else
   }//updateUncommitted

   /* This function returns the best task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickNeighborTask(AONTaskList tasks){
      return pickBestNeighborTask(tasks);
   }//pickNeighborTask

   /* This function returns the best task which this agent is needed for
*/
   public AONTask pickTask(AONTaskList tasks){
      return pickBestTask(tasks);
   }//pickTask

   /* This function returns whether the agent has chosen to drop its
*/
```

```
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      boolean dropTask = false;
      if(getUnCommittedNeighborCount() < task.unFilledSkillsCount()){
         dropTask = true;
      }//if
      return dropTask;
   }//chooseToDropTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "PerformanceStratImpatient_" + super.toString() + "\n";
      return value;
   }//toString
}//class PerformanceStratImpatientAgent

------------------PerformanceStratImpatientAgent.java--------------------------
-

--------------------------------PieChart.java-----------------------------------
-

import java.awt.*;
import java.util.Vector;
import java.io.IOException;
import java.io.File;
import java.awt.image.*;
import javax.imageio.ImageIO;

/**
 * PieChart Class
 *
 *
 */
public class PieChart implements Runnable{
   String m_inputFilename;   /*
*/
   CSVFile m_countFile;      /*
*/

   Vector<String> m_outputDirectories;
   Vector<String> m_outputFilenames;
   Vector<String> m_chartTitles;
   Vector<Integer> m_valueRows;

//    String m_outputDirectory;
                             /*
*/
//    String m_outputFilename;
                             /*
*/
   String m_chartTitle;      /*
*/
   int m_valuesRow;          /*
*/

   boolean m_drawTitle;      /*
*/
   boolean m_drawCatagoryTags;
                             /*
*/
```

```
   int m_headerRow;          /*
*/
   int m_startColumn;        /*
*/
   float m_sumValues;        /*
*/
   int m_dimension;          /*
*/

   int m_imageType;//must be a valid BufferedImage constant value
                             /*
*/

   /* constructor
*/
   public PieChart(){
      m_inputFilename = "";

      m_outputDirectories = new Vector();
      m_outputFilenames = new Vector();
      m_chartTitles = new Vector();
      m_valueRows = new Vector();
//       m_outputDirectory = "";
//       m_outputFilename = "";
       m_chartTitle = "";
       m_valuesRow = 3;

      m_drawTitle = false;
      m_drawCatagoryTags = true;
      m_headerRow = 2;
      m_startColumn = 1;
      m_sumValues = 0.0f;

      m_imageType = BufferedImage.TYPE_INT_RGB;
   }//constructor

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem
```

```
   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   public static void main(String args[]){
      PieChart pie = new PieChart();

      for(int i = 0; i < args.length; i++){
         pie.processArg(i,args);
      }//for

      pie.run();
   }//main

   private boolean processArg(int i, String args[]){
      try{
         if(args[i].compareTo("-input") == 0){
            m_inputFilename = args[i+1];
         }//if
      }catch(Exception ex){
         System.out.println("Invalid argument");
         System.out.println("Usage: PieChart -input <text>[the filename of the
xml file containing the pie chart parameters]");
         return false;
      }//catch
      return true;
   }//processArg

   public void run(){
      if(m_inputFilename != ""){
         readInputFile();

         for(int i = 0; i < getLength(m_outputDirectories); i++){
            String outputDirectory = (String)getItem(i,m_outputDirectories);
            String outputFilename = (String)getItem(i,m_outputFilenames);
            m_chartTitle = (String)getItem(i,m_chartTitles);
            m_valuesRow = ((Integer)getItem(i,m_valueRows)).intValue();
            createPieChart(outputDirectory,outputFilename);
         }//for
      }//if
   }//run

   public void setTitle(String chartTitle){
      m_drawTitle = true;
      m_chartTitle = chartTitle;
   }//setTitle

   public void setDrawCatagoryTags(boolean value){
      m_drawCatagoryTags = value;
   }//setDrawCatagoryTags

   public void setHeaderRow(int headerRow){
      m_headerRow = headerRow;
   }//setHeaderRow

   public void setValuesRow(int valuesRow){
      m_valuesRow = valuesRow;
   }//setValuesRow

   public void setStartColumn(int startColumn){
```

```
      m_startColumn = startColumn;
   }//setStartColumn

   public void setGrayscale(boolean value){
      if(value){
         m_imageType = BufferedImage.TYPE_BYTE_GRAY;
      }else{
         m_imageType = BufferedImage.TYPE_INT_RGB;
      }//else
   }//setImageType

   public void createPieChart(String outputDirectory, String outputFilename,
CSVFile CountResults){
      int dimension = 600;
      if(m_countFile != null){
         m_countFile.close();
      }//if
      m_countFile = CountResults;
      BufferedImage bufferedImage = bufferedImage = new
BufferedImage(dimension,dimension,m_imageType);
      drawPieChart((Graphics)bufferedImage.getGraphics(),dimension);
      try{
//          String names[] = ImageIO.getWriterFormatNames();
//          for(int i = 0; i < names.length; i++){
//             System.out.print(" " + names[i]);
//          }//for
         if(checkDirectory(outputDirectory)){
            outputFilename = outputDirectory + "/" + outputFilename + ".jpg";
         }else{
            outputFilename = outputFilename + ".jpg";
         }//else
         ImageIO.write(((RenderedImage)bufferedImage), "JPEG", new
File(outputFilename));
      }catch(IOException ex){
         System.out.println("Error creating pie chart: " + ex);
      }//catch
   }//createPieChart

   private void createPieChart(String outputDirectory, String outputFilename){
      int dimension = 600;
      BufferedImage bufferedImage = bufferedImage = new
BufferedImage(dimension,dimension,m_imageType);
      drawPieChart((Graphics)bufferedImage.getGraphics(),dimension);
      try{
//          String names[] = ImageIO.getWriterFormatNames();
//          for(int i = 0; i < names.length; i++){
//             System.out.print(" " + names[i]);
//          }//for
         if(checkDirectory(outputDirectory)){
            outputFilename = outputDirectory + "/" + outputFilename + ".jpg";
         }else{
            outputFilename = outputFilename + ".jpg";
         }//else
         ImageIO.write(((RenderedImage)bufferedImage), "JPEG", new
File(outputFilename));
      }catch(IOException ex){
         System.out.println("Error creating pie chart: " + ex);
      }//catch
   }//createPieChart

   private void drawPieChart(Graphics g, int dimension){
      g.setColor(Color.white);
      g.fillRect(0,0,dimension,dimension);
```

```
      m_sumValues = 0.0f;
      m_dimension = dimension;

      computeValuesSum();

      if(m_drawCatagoryTags){
         drawCatagoryTags(g);
      }//if
      drawPieCatagories(g);
      if(m_drawTitle){
         drawCenteredTitle(g);
      }//if
   }//drawPieChart

   private void computeValuesSum(){
      int column = m_startColumn;
      if(m_countFile == null){
         System.out.println("m_countFile is null.");
      }//if
      String iter = m_countFile.getItem(m_valuesRow,0);
      String title = m_countFile.getItem(m_headerRow,column);
      String item = m_countFile.getItem(m_valuesRow,column);
      //System.out.println("PieChart.ComputeValuesSum()\n  Iteration: " + iter
+ " Column: " + column + " HeaderRow: " + m_headerRow + " ValuesRow: " +
m_valuesRow + " Title: " + title + " Item: " + item);
      while(item != null){
         float value = 0.0f;
         try{
            value = Float.parseFloat(item);
            m_sumValues = m_sumValues + value;
         }catch(Exception ex){
            System.out.println("Error getting value: " + ex);
         }//catch
         column++;
         title = m_countFile.getItem(m_headerRow,column);
         item = m_countFile.getItem(m_valuesRow,column);
         //System.out.println("  Column: " + column + " HeaderRow: " +
m_headerRow + " ValuesRow: " + m_valuesRow + " Title: " + title + " Item: " +
item);
      }//while
   }//computeValuesSum

   private void drawPieCatagories(Graphics g){
      Color colors[] = {Color.red, Color.blue, Color.green, Color.yellow,
Color.cyan};
      int index = 0, sumArc = 0;
      int column = m_startColumn;
      String item = m_countFile.getItem(m_valuesRow,column);
      while(item != null){
         float value = 0.0f;
         try{
            value = Float.parseFloat(item);
            if(value > 0.0f){
               int startAngle = sumArc, angle =
(int)((float)360.0f*value/m_sumValues);
               int catagoryAngle = startAngle + (angle/2);
               int adjacent =
(int)(Math.cos(degreesToRadians(catagoryAngle))*m_dimension/3.0f), opposite =
(int)(Math.sin(degreesToRadians(catagoryAngle))*m_dimension/3.0f);
               int x = (m_dimension/2) + adjacent, y = (m_dimension/2) -
opposite;
               if(index == 0){
                  startAngle = startAngle - 5;
```

```
                  angle = angle + 5;
               }//if
               g.setColor(Color.black);
               Color c = colors[index%colors.length];
               g.setColor(c);

g.fillArc(m_dimension/4,m_dimension/4,m_dimension/2,m_dimension/2,startAngle,an
gle);
               g.setColor(Color.black);

g.drawArc(m_dimension/4,m_dimension/4,m_dimension/2,m_dimension/2,startAngle,an
gle);
               if(index == 0){
                  angle = angle - 5;
               }//if
               sumArc = sumArc + angle;
            }//if
            System.out.println("Value=" + value + " sumArc=" + sumArc);
         }catch(Exception ex){
            System.out.print("Error getting value: ");
            ex.printStackTrace();
         }//catch
         column++;
         index++;
         item = m_countFile.getItem(m_valuesRow,column);
      }//while
   }//drawPieCatagories

   private void drawCatagoryTags(Graphics g){
      int sumArc = 0, column = m_startColumn, lastX = 0, lastY = 0;
      String title = m_countFile.getItem(m_headerRow,column);
      String item = m_countFile.getItem(m_valuesRow,column);
      g.setFont(new Font("Dialog",Font.PLAIN,12));
      while(item != null){
         float value = 0.0f;
         try{
            value = Float.parseFloat(item);
            if(value > 0.0f){
               int startAngle = sumArc, angle =
(int)((float)360.0f*value/m_sumValues);
               int tagAngle = startAngle + (angle/2);
               double cosine = Math.cos(degreesToRadians(tagAngle));
               double sine = Math.sin(degreesToRadians(tagAngle));
               int adjacent = (int)(cosine*m_dimension/4.0f), opposite =
(int)(sine*m_dimension/4.0f);
               int startX = (m_dimension/2) + adjacent, startY =
(m_dimension/2) - opposite;
               adjacent = (int)(cosine*m_dimension/3.0f);
               opposite = (int)(sine*m_dimension/3.0f);
               int tagX = (m_dimension/2) + adjacent, tagY = (m_dimension/2) -
opposite;
               System.out.println("INITIAL: tagX=" + tagX + " tagY=" + tagY + "
lastY=" + lastY + " " );
               if(Math.sqrt((tagX-lastX)*(tagX-lastX) + (tagY-lastY)*(tagY-
lastY)) <= 14){
                  if(angle < 90 || angle > 270){
                     tagY = lastY - 14;
                  }else{
                     tagY = lastY + 14;
                  }//else
               }//if
               g.setColor(Color.gray);
               g.drawLine(startX,startY,tagX,tagY);
```

```
               g.setColor(Color.black);
               String percentString =
String.format("%.1f",(100.0f*value/m_sumValues));
               String tagString = percentString + "% " + title;
               tagX = adjustTagX(tagX, tagAngle, g.getFont(), tagString);
               g.drawString(tagString,tagX,tagY);

               //g.drawRect(tagX,tagY-
12,getTextPixelWidth(tagString,g.getFont()),12);
               sumArc = sumArc + angle;

               System.out.print("FINAL: tagX=" + tagX + " tagY=" + tagY + "
lastY=" + lastY + " " );
               lastX = tagX;
               lastY = tagY;
            }//if
            System.out.println("Title=" + title + " Value=" + value + "
sumArc=" + sumArc);
         }catch(Exception ex){
            System.out.print("Error getting value: ");
            ex.printStackTrace();
         }//catch
         column++;
         title = m_countFile.getItem(m_headerRow,column);
         item = m_countFile.getItem(m_valuesRow,column);
      }//while
   }//drawCatagoryTags

   private void drawCenteredTitle(Graphics g){
      int titleWidth = getTextPixelWidth(m_chartTitle,g.getFont());
      int x = (m_dimension/2) - (titleWidth/2), y = m_dimension/8;
      g.setFont(new Font("Times New Roman",Font.BOLD,14));
      g.drawString(m_chartTitle, x, y);
   }//drawCenteredTitle

   private int adjustTagX(int x, int tagAngle, Font tagFont, String tagString){
      int newX = x;
      if(tagAngle > 90 && tagAngle < 270){
         newX = x - getTextPixelWidth(tagString,tagFont);
      }//if

      if(newX + getTextPixelWidth(tagString,tagFont) > m_dimension){
         newX = m_dimension - getTextPixelWidth(tagString,tagFont);
      }else if(newX < 0){
         newX = 0;
      }//if

      return newX;
   }//adjustTagX

   private int getTextPixelWidth(String testString, Font testFont){
      int characterWidth = (int)(testFont.getSize2D()/1.6f); //7;
      //System.out.println("Font name = " + testFont.getName() + " Font size =
" + testFont.getSize2D() + " characterWidth = " + characterWidth);
      return characterWidth*testString.length();
   }//getTextPixelWidth

   private float degreesToRadians(float degrees){
      return (float)(2.0f*Math.PI/360.0f)*degrees;
   }//degreesToRadians

   private boolean checkDirectory(String outputDirectory){
      File outDirectory = new File(outputDirectory);
```

```
      if(!outDirectory.isDirectory()){
         return outDirectory.mkdir();
      }//if
      return outDirectory.exists();
   }//checkDirectory

   private void readInputFile(){
      System.out.println("readInputFile");
      AONXMLReader xmlFile = new AONXMLReader();
      if(!xmlFile.openXML(m_inputFilename)){
         System.out.println("Bad XML File: " + m_inputFilename + " Using
default pie chart settings");
         return;
      }//if

      XMLElement root = xmlFile.getRoot();
      String attributeName = "";
      String value = "";
      try{
         attributeName = "DrawTags";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_drawCatagoryTags =
parseBoolean(value);

         attributeName = "HeaderRow";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_headerRow = Integer.parseInt(value);

         attributeName = "StartColumn";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_startColumn =
Integer.parseInt(value);

         boolean grayScale = false;
         attributeName = "GrayScale";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)grayScale = parseBoolean(value);
         if(grayScale){
            m_imageType = BufferedImage.TYPE_BYTE_GRAY;
         }else{
            m_imageType = BufferedImage.TYPE_INT_RGB;
         }//else

         attributeName = "InputFilename";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null){
            if(m_countFile != null){
               m_countFile.close();
            }//if
            m_countFile = new CSVFile(value);
            if(m_countFile == null){
               System.out.println("Null pointer to m_countFile!");
            }//if
         }//if

         int chartCount = root.getNumElements();
         for(int i = 0; i < chartCount; i++){
            XMLElement xmlChart = root.getElement(i);
            attributeName = "Title";
            value = xmlChart.getAttribute(attributeName);
            if(value != "" && value != null){
               m_drawTitle = true;
               m_chartTitles.add(value);
```

```
            }else{
               m_chartTitles.add("");
            }//else

            attributeName = "ValuesRow";
            value = xmlChart.getAttribute(attributeName);
            if(value != "" && value != null){
               m_valueRows.add(Integer.parseInt(value));
            }else{
               m_valueRows.add(1);
            }//else

            attributeName = "OutputDirectory";
            value = xmlChart.getAttribute(attributeName);
            if(value != "" && value != null){
               m_outputDirectories.add(value);
            }else{
               m_outputDirectories.add("");
            }//else

            attributeName = "OutputFilename";
            value = xmlChart.getAttribute(attributeName);
            if(value != "" && value != null){
               m_outputFilenames.add(value);
            }else{
               m_outputFilenames.add("");
            }//else
         }//for
      }catch(Exception ex){
         System.out.print("Error reading piechart xml: ");
         ex.printStackTrace();
      }//catch
   }//readInputFile

   private boolean parseBoolean(String value){
      if(value.compareToIgnoreCase("true") == 0){
         return true;
      }else{
         return false;
      }//else
   }//parseBoolean
}//class PieChart

--------------------------------PieChart.java-----------------------------------
-

----------------------------PriorityQueue.java--------------------------------
-

import java.util.*;

/**
 * PriorityQueue Class
 *
 *
 */
public class PriorityQueue{
   private boolean m_ascending;
                              /*
*/
   private Vector<QueueItem> m_queue;
```

```
   /* constructor
*/
   public PriorityQueue(){
      m_ascending = true;
      m_queue = new Vector();
   }//constructor

   /* constructor
*/
   public PriorityQueue(boolean ascending){
      m_ascending = ascending;
      m_queue = new Vector();
   }//constructor

   public void add(Object item, int score){
      QueueItem newItem = new QueueItem(item,score);

      boolean done = false;
      for(int i = 0; i < m_queue.size() && !done; i++){
         QueueItem current = m_queue.elementAt(i);
         if(m_ascending){
            if(current.greaterThan(newItem)){
               done = true;
            }//if
         }else{
            if(current.lessThan(newItem)){
               done = true;
            }//if
         }//else
         if(done){
            m_queue.insertElementAt(newItem,i);
         }//if
      }//for

      if(!done){
         m_queue.add(newItem);
      }//if
   }//add

   public Object remove(){
      QueueItem QueueItem = m_queue.elementAt(0);
      m_queue.removeElementAt(0);
      return QueueItem.getItem();
   }//remove

   public int size(){
      return m_queue.size();
   }//size

   public boolean isEmpty(){
      return (size() <= 0);
   }//isEmpty

   public String toString(){
      return m_queue.toString();
   }//toString
}//PriorityQueue
```

----------PriorityQueue.java----------
-

----------QDParser.java----------
-

```
import java.io.*;
import java.util.*;

/**
 * QDParser Class
 *   Quick and Dirty xml parser.  This parser is, like the SAX parser,
 *   an event based parser, but with much less functionality.
 *
 */
public class QDParser {
  private static int popMode(Stack st) {
    if(!st.empty())
      return ((Integer)st.pop()).intValue();
    else
      return PRE;
  }
  private final static int
    TEXT = 1,
    ENTITY = 2,
    OPEN_TAG = 3,
    CLOSE_TAG = 4,
    START_TAG = 5,
    ATTRIBUTE_LVALUE = 6,
    ATTRIBUTE_EQUAL = 9,
    ATTRIBUTE_RVALUE = 10,
    QUOTE = 7,
    IN_TAG = 8,
    SINGLE_TAG = 12,
    COMMENT = 13,
    DONE = 11,
    DOCTYPE = 14,
    PRE = 15,
    CDATA = 16;
  public static void parse(DocHandler doc,Reader r) throws Exception {
    Stack st = new Stack();
    int depth = 0;
    int mode = PRE;
    int c = 0;
    int quotec = '"';
    depth = 0;
    StringBuffer sb = new StringBuffer();
    StringBuffer etag = new StringBuffer();
    String tagName = null;
    String lvalue = null;
    String rvalue = null;
    Hashtable attrs = null;
    st = new Stack();
    doc.startDocument();
    int line=1, col=0;
    boolean eol = false;
    while((c = r.read()) != -1) {

      // We need to map \r, \r\n, and \n to \n
      // See XML spec section 2.11
      if(c == '\n' && eol) {
        eol = false;
   continue;
      } else if(eol) {
        eol = false;
      } else if(c == '\n') {
        line++;
   col=0;
```

```
   } else if(c == '\r') {
     eol = true;
c = '\n';
line++;
col=0;
   } else {
     col++;
   }

   if(mode == DONE) {
doc.endDocument();
     return;

   // We are between tags collecting text.
   } else if(mode == TEXT) {
     if(c == '<') {
  st.push(new Integer(mode));
  mode = START_TAG;
  if(sb.length() > 0) {
    doc.text(sb.toString());
    sb.setLength(0);
  }
     } else if(c == '&') {
  st.push(new Integer(mode));
  mode = ENTITY;
  etag.setLength(0);
} else
  sb.append((char)c);

   // we are processing a closing tag: e.g. </foo>
   } else if(mode == CLOSE_TAG) {
     if(c == '>') {
  mode = popMode(st);
  tagName = sb.toString();
  sb.setLength(0);
  depth--;
  if(depth==0)
    mode = DONE;
  doc.endElement(tagName);
} else {
  sb.append((char)c);
}

   // we are processing CDATA
   } else if(mode == CDATA) {
     if(c == '>'
&& sb.toString().endsWith("]]")) {
  sb.setLength(sb.length()-2);
  doc.text(sb.toString());
  sb.setLength(0);
  mode = popMode(st);
} else
  sb.append((char)c);

   // we are processing a comment.  We are inside
   // the <!-- .... --> looking for the -->.
   } else if(mode == COMMENT) {
     if(c == '>'
&& sb.toString().endsWith("--")) {
  sb.setLength(0);
  mode = popMode(st);
} else
  sb.append((char)c);
```

```
   // We are outside the root tag element
   } else if(mode == PRE) {
if(c == '<') {
  mode = TEXT;
  st.push(new Integer(mode));
  mode = START_TAG;
     }

   // We are inside one of these <? ... ?>
   // or one of these <!DOCTYPE ... >
   } else if(mode == DOCTYPE) {
if(c == '>') {
  mode = popMode(st);
  if(mode == TEXT) mode = PRE;
}

   // we have just seen a < and
   // are wondering what we are looking at
   // <foo>, </foo>, <!-- ... --->, etc.
   } else if(mode == START_TAG) {
     mode = popMode(st);
if(c == '/') {
  st.push(new Integer(mode));
  mode = CLOSE_TAG;
} else if (c == '?') {
  mode = DOCTYPE;
     } else {
  st.push(new Integer(mode));
  mode = OPEN_TAG;
  tagName = null;
  attrs = new Hashtable();
  sb.append((char)c);
     }

   // we are processing an entity, e.g. &lt;, &#187;, etc.
   } else if(mode == ENTITY) {
     if(c == ';') {
  mode = popMode(st);
  String cent = etag.toString();
  etag.setLength(0);
  if(cent.equals("lt"))
    sb.append('<');
  else if(cent.equals("gt"))
    sb.append('>');
  else if(cent.equals("amp"))
    sb.append('&');
  else if(cent.equals("quot"))
    sb.append('"');
  else if(cent.equals("apos"))
    sb.append('\'');
  // Could parse hex entities if we wanted to
  //else if(cent.startsWith("#x"))
    //sb.append((char)Integer.parseInt(cent.substring(2),16));
  else if(cent.startsWith("#"))
    sb.append((char)Integer.parseInt(cent.substring(1)));
  // Insert custom entity definitions here
  else
    exc("Unknown entity: &"+cent+";",line,col);
} else {
  etag.append((char)c);
}
```

```
   // we have just seen something like this:
   // <foo a="b"/
   // and are looking for the final >.
   } else if(mode == SINGLE_TAG) {
if(tagName == null)
  tagName = sb.toString();
     if(c != '>')
  exc("Expected > for tag: <"+tagName+"/>",line,col);
doc.startElement(tagName,attrs);
doc.endElement(tagName);
if(depth==0) {
  doc.endDocument();
  return;
}
sb.setLength(0);
attrs = new Hashtable();
tagName = null;
mode = popMode(st);

   // we are processing something
   // like this <foo ... >.  It could
   // still be a <!-- ... --> or something.
   } else if(mode == OPEN_TAG) {
if(c == '>') {
  if(tagName == null)
        tagName = sb.toString();
  sb.setLength(0);
  depth++;
  doc.startElement(tagName,attrs);
  tagName = null;
  attrs = new Hashtable();
  mode = popMode(st);
} else if(c == '/') {
  mode = SINGLE_TAG;
} else if(c == '-' && sb.toString().equals("!-")) {
  mode = COMMENT;
} else if(c == '[' && sb.toString().equals("![CDATA")) {
  mode = CDATA;
  sb.setLength(0);
} else if(c == 'E' && sb.toString().equals("!DOCTYP")) {
  sb.setLength(0);
  mode = DOCTYPE;
} else if(Character.isWhitespace((char)c)) {
  tagName = sb.toString();
  sb.setLength(0);
  mode = IN_TAG;
} else {
  sb.append((char)c);
}

   // We are processing the quoted right-hand side
   // of an element's attribute.
   } else if(mode == QUOTE) {
     if(c == quotec) {
  rvalue = sb.toString();
  sb.setLength(0);
  attrs.put(lvalue,rvalue);
  mode = IN_TAG;
// See section the XML spec, section 3.3.3
// on normalization processing.
} else if(" \r\n\u0009".indexOf(c)>=0) {
  sb.append(' ');
} else if(c == '&') {
```

```
    st.push(new Integer(mode));
    mode = ENTITY;
    etag.setLength(0);
  } else {
    sb.append((char)c);
  }

     } else if(mode == ATTRIBUTE_RVALUE) {
       if(c == '"' || c == '\'') {
    quotec = c;
    mode = QUOTE;
  } else if(Character.isWhitespace((char)c)) {
    ;
  } else {
    exc("Error in attribute processing",line,col);
  }

     } else if(mode == ATTRIBUTE_LVALUE) {
       if(Character.isWhitespace((char)c)) {
    lvalue = sb.toString();
    sb.setLength(0);
    mode = ATTRIBUTE_EQUAL;
  } else if(c == '=') {
    lvalue = sb.toString();
    sb.setLength(0);
    mode = ATTRIBUTE_RVALUE;
  } else {
    sb.append((char)c);
  }

     } else if(mode == ATTRIBUTE_EQUAL) {
       if(c == '=') {
    mode = ATTRIBUTE_RVALUE;
  } else if(Character.isWhitespace((char)c)) {
    ;
  } else {
    exc("Error in attribute processing.",line,col);
  }

     } else if(mode == IN_TAG) {
  if(c == '>') {
    mode = popMode(st);
    doc.startElement(tagName,attrs);
    depth++;
    tagName = null;
    attrs = new Hashtable();
  } else if(c == '/') {
    mode = SINGLE_TAG;
       } else if(Character.isWhitespace((char)c)) {
    ;
  } else {
    mode = ATTRIBUTE_LVALUE;
    sb.append((char)c);
  }
     }
   }
   if(mode == DONE)
     doc.endDocument();
   else
     exc("missing end tag",line,col);
}
private static void exc(String s,int line,int col)
  throws Exception
```

```
  {
    throw new Exception(s+" near line "+line+", column "+col);
  }
}

------------------------------QDParser.java-------------------------------------
-

------------------------------QueueItem.java-------------------------------------
-

/**
 * QueueItem Class
 *
 *
 */
public class QueueItem{
   private int m_score;      /*
*/
   private Object m_item;    /*
*/

   /* constructor
*/
   public QueueItem(Object i, int s){
      m_score = s;
      m_item = i;
   }//constructor

   public int getScore(){
      return m_score;
   }//getScore

   public Object getItem(){
      return m_item;
   }//getItem

   public boolean lessThan(QueueItem item){
      return m_score < item.getScore();
   }//lessThan

   public boolean greaterThan(QueueItem item){
      return m_score > item.getScore();
   }//lessThan


   public String toString(){
      return m_item + ":" + m_score;
   }//toString
}//QueueItem

------------------------------QueueItem.java-------------------------------------
-

------------------------------ReportAON2.java-------------------------------------
-

import java.util.*;
import java.io.IOException;
import java.io.BufferedWriter;
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.FileReader;
```

```
import java.io.File;

/**
 * ReportAON2 Class
 *
 *
 */
public class ReportAON2 implements Runnable{
   boolean m_insertFirstColumn;
                              /*
*/

   String m_inputFilename;   /*
*/
   String m_outputDirectory; /*
*/
   String m_reportDescription;
                              /*
*/
   int m_numberOfInputFiles; /*
*/

   int m_numberOfTests;      /*
*/
   Vector<String> m_testNames;
   Vector<String> m_outputDirectories;
   Vector<String> m_descriptions;

   Vector<String> m_plotFileNames;

   int m_averageColumnIndex; /*
*/
   int m_standardErrorColumnIndex;
                              /*
*/
   int m_improvementColumnIndex;
                              /*
*/

   int m_minAdaptTimeStep;   /*
*/
   boolean m_gnuplotOnly;    /*
*/
   Vector<String> m_agentTypes;
   int m_numberOfSkills;     /*
*/

   /* constructor
*/
   public ReportAON2(){
      m_insertFirstColumn = true;

      m_inputFilename = "";
      m_outputDirectory = "";
      m_reportDescription = "";
      m_numberOfInputFiles = 3;

      m_testNames = new Vector();
      m_outputDirectories = new Vector();
      m_descriptions = new Vector();

      m_plotFileNames = new Vector();
```

```
      m_gnuplotOnly = false;
      m_agentTypes = new Vector();
   }//constructor

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   public void setInputFilename(String inputFilename){
      m_inputFilename = inputFilename;
   }//setInputFilename

   public void setCountInput(int countInput){
      m_numberOfInputFiles = countInput;
   }//setCountInput

   public static void main(String args[]){
      ReportAON2 report = new ReportAON2();

      for(int i = 0; i < args.length; i++){
         report.processArg(i,args);
      }//for

      report.run();
   }//main

   private boolean processArg(int i, String args[]){
      try{
         if(args[i].compareTo("-input") == 0){
            m_inputFilename = args[i+1];
         }else if(args[i].compareTo("-output_directory") == 0){
            m_outputDirectory = args[i+1];
         }else if(args[i].compareTo("-count_input") == 0){
```

```
            m_numberOfInputFiles = Integer.parseInt(args[i+1]);
         }else if(args[i].compareTo("-gnuplot_only") == 0){
            m_gnuplotOnly = true;
         }else{
            System.out.println("Invalid argument: " + args[i]);
            System.out.println("Usage: ReportAON [-output_directory <filename>]
[-count_input <number>] [-description <string>] [-gnuplot_only]");
         }//else
      }catch(Exception ex){
         System.out.println("Invalid argument");
         System.out.println("Usage: ReportAON [-output_directory <filename>] [-
count_input <number>] [-description <string>] [-gnuplot_only]");
         return false;
      }//catch
      return true;
   }//processArg

   public void run(){
      if(m_inputFilename != ""){
         readInputFile();
      }//if

      if(m_gnuplotOnly){
         createGnuPlot();
      }else{
         processReport();
      }//else
   }//run

   private void createGnuPlot(){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile OptimalPerformanceResults = new CSVFile(aonDirectoryString +
"/OptimalPerformanceResults" + m_outputDirectory + ".csv");
      CSVFile PerformanceResults = new CSVFile(aonDirectoryString +
"/PerformanceResults" + m_outputDirectory + ".csv");
      CSVFile PerformanceErrorResults = new CSVFile(aonDirectoryString +
"/PerformanceErrorResults" + m_outputDirectory + ".csv");
      CSVFile PerformanceImprovementResults = new CSVFile(aonDirectoryString +
"/PerformanceImprovementResults" + m_outputDirectory + ".csv");
      CSVFile EfficiencyResults = new CSVFile(aonDirectoryString +
"/EfficiencyResults" + m_outputDirectory + ".csv");
      CSVFile EfficiencyErrorResults = new CSVFile(aonDirectoryString +
"/EfficiencyErrorResults" + m_outputDirectory + ".csv");
      CSVFile MinDegreeResults = new CSVFile(aonDirectoryString +
"/MinDegreeResults" + m_outputDirectory + ".csv");
      CSVFile MaxDegreeResults = new CSVFile(aonDirectoryString +
"/MaxDegreeResults" + m_outputDirectory + ".csv");
      CSVFile DegreeVarianceResults = new CSVFile(aonDirectoryString +
"/DegreeVarianceResults" + m_outputDirectory + ".csv");
      CSVFile AverageDegreeResults = new CSVFile(aonDirectoryString +
"/AverageDegreeResults" + m_outputDirectory + ".csv");
      CSVFile AverageDensityResults = new CSVFile(aonDirectoryString +
"/AverageDensityResults" + m_outputDirectory + ".csv");
      CSVFile ChangedEdgesResults = new CSVFile(aonDirectoryString +
"/ChangedEdgesResults" + m_outputDirectory + ".csv");
      CSVFile TimeSpentResults = new CSVFile(aonDirectoryString +
"/TimeSpentResults" + m_outputDirectory + ".csv");
      CSVFile AgentTypeCountResults = new CSVFile(aonDirectoryString +
"/AgentCountResults" + m_outputDirectory + ".csv");

      writePlotOptimalPerformanceFile(OptimalPerformanceResults);
      writePlotPerformanceFile(PerformanceResults);
      writePlotPerformanceYErrorFile(PerformanceResults);
```

```
      writePlotPerformanceErrorFile(PerformanceErrorResults);
      writePlotPerformanceImprovementFile(PerformanceImprovementResults);
      writePlotEfficiencyFile(EfficiencyResults);
      writePlotEfficiencyYErrorFile(EfficiencyResults);
      writePlotEfficiencyErrorFile(EfficiencyErrorResults);
      writePlotMinDegreeFile(MinDegreeResults);
      writePlotMaxDegreeFile(MaxDegreeResults);
      writePlotDegreeVarianceFile(DegreeVarianceResults);
      writePlotAverageDegreeFile(AverageDegreeResults);
      writePlotAverageDensityFile(AverageDensityResults);
      writePlotChangedEdgesFile(ChangedEdgesResults);
      writeTimeSpentFile(TimeSpentResults);
      writeAgentCountFile(AgentTypeCountResults);
      writePlotAllFile();
   }//createGnuPlot

   public void processReport(){
      System.out.println("Process ReportAON2: \n    " + m_testNames + "\n    "
+ m_outputDirectories + "\n    " + m_descriptions);

      m_averageColumnIndex = m_numberOfInputFiles + 1;
      m_standardErrorColumnIndex = m_numberOfInputFiles + 2;
      m_improvementColumnIndex = m_numberOfInputFiles + 3;

      //These vectors contain sets of files output by ReportAON
      // for each of the adaptation tests that have been run.
      Vector TempOptimalPerformanceResults = new Vector();
      Vector TempPerformanceResults = new Vector();
      Vector TempEfficiencyResults = new Vector();
      Vector TempMinDegreeResults = new Vector();
      Vector TempMaxDegreeResults = new Vector();
      Vector TempDegreeVarianceResults = new Vector();
      Vector TempAverageDegreeResults = new Vector();
      Vector TempAverageDensityResults = new Vector();
      Vector TempChangedEdgesResults = new Vector();
      Vector TempTimeWaitingResults = new Vector();
      Vector TempTimeRewiringResults = new Vector();
      Vector TempTimeProposingResults = new Vector();
      Vector TempTimeJoiningResults = new Vector();
      Vector TempTimeWorkingResults = new Vector();
      Vector TempChangedAgentsResults = new Vector();
      Vector TempTaskDistanceResults = new Vector();
      Vector TempAgentTypeCountResults = new Vector();
      Vector TempPendingFailureResults = new Vector();
      Vector TempSkillFailureResults = new Vector();
      Vector TempDistanceFailureResults = new Vector();
      Vector TempTaskCompleteResults = new Vector();

      Vector TempTaskClustersResults = new Vector();
      Vector TempTaskCountResults = new Vector();
      Vector TempDistanceTravelledResults = new Vector();

      Vector TempJoinedImpossibleResults = new Vector();
      Vector TempProposedImpossibleResults = new Vector();
      Vector TempRepeatTaskResults = new Vector();
      Vector TempMinSkillShortageResults = new Vector();
      Vector TempMaxSkillShortageResults = new Vector();
      Vector TempAverageSkillShortageResults = new Vector();
      Vector TempAliveAgentsResults = new Vector();

      Vector TempSkillCountResults = new Vector();
      Vector TempSkillFailureCountResults = new Vector();
      Vector TempSkillDemandCountResults = new Vector();
```

```
      Vector headerValues = new Vector();
      headerValues.add(" ");

      for(int i = 0; i < m_numberOfTests; i++){
         String testName = (String)getItem(i,m_testNames);
         String directoryString = (String)getItem(i,m_outputDirectories);
         String description = (String)getItem(i,m_descriptions);
         TempOptimalPerformanceResults.add(new
CSVFile(directoryString+"/ReportAONOptimalPerformance_" + description +
".csv"));
         TempPerformanceResults.add(new
CSVFile(directoryString+"/ReportAONPerformance_" + description + ".csv"));
         TempEfficiencyResults.add(new
CSVFile(directoryString+"/ReportAONEfficiency_" + description + ".csv"));
         TempMinDegreeResults.add(new
CSVFile(directoryString+"/ReportAONMinDegree_" + description + ".csv"));
         TempMaxDegreeResults.add(new
CSVFile(directoryString+"/ReportAONMaxDegree_" + description + ".csv"));
         TempDegreeVarianceResults.add(new
CSVFile(directoryString+"/ReportAONDegreeVariance_" + description + ".csv"));
         TempAverageDegreeResults.add(new
CSVFile(directoryString+"/ReportAONAverageDegree_" + description + ".csv"));
         TempAverageDensityResults.add(new
CSVFile(directoryString+"/ReportAONAverageDensity_" + description + ".csv"));
         TempChangedEdgesResults.add(new
CSVFile(directoryString+"/ReportAONChangedEdges_" + description + ".csv"));
         TempTimeWaitingResults.add(new
CSVFile(directoryString+"/ReportAONTimeWaiting_" + description + ".csv"));
         TempTimeRewiringResults.add(new
CSVFile(directoryString+"/ReportAONTimeRewiring_" + description + ".csv"));
         TempTimeProposingResults.add(new
CSVFile(directoryString+"/ReportAONTimeProposing_" + description + ".csv"));
         TempTimeJoiningResults.add(new
CSVFile(directoryString+"/ReportAONTimeJoining_" + description + ".csv"));
         TempTimeWorkingResults.add(new
CSVFile(directoryString+"/ReportAONTimeWorking_" + description + ".csv"));
         TempChangedAgentsResults.add(new
CSVFile(directoryString+"/ReportAONChangedAgents_" + description + ".csv"));
         TempTaskDistanceResults.add(new
CSVFile(directoryString+"/ReportAONTaskDistance_" + description + ".csv"));
         TempPendingFailureResults.add(new
CSVFile(directoryString+"/ReportAONPendingFailures_" + description + ".csv"));
         TempSkillFailureResults.add(new
CSVFile(directoryString+"/ReportAONSkillFailures_" + description + ".csv"));
         TempDistanceFailureResults.add(new
CSVFile(directoryString+"/ReportAONDistanceFailures_" + description + ".csv"));
         TempTaskCompleteResults.add(new
CSVFile(directoryString+"/ReportAONTasksCompleted_" + description + ".csv"));

         TempTaskClustersResults.add(new
CSVFile(directoryString+"/ReportAONTaskClusters_" + description + ".csv"));
         TempTaskCountResults.add(new
CSVFile(directoryString+"/ReportAONTaskCount_" + description + ".csv"));
         TempDistanceTravelledResults.add(new
CSVFile(directoryString+"/ReportAONDistanceTravelled_" + description +
".csv"));

         TempJoinedImpossibleResults.add(new
CSVFile(directoryString+"/ReportAONJoinedImpossible_" + description + ".csv"));
         TempProposedImpossibleResults.add(new
CSVFile(directoryString+"/ReportAONProposedImpossible_" + description +
".csv"));
```

```
         TempRepeatTaskResults.add(new
CSVFile(directoryString+"/ReportAONRepeatTask_" + description + ".csv"));
         TempMinSkillShortageResults.add(new
CSVFile(directoryString+"/ReportAONMinSkillShortage_" + description + ".csv"));
         TempMaxSkillShortageResults.add(new
CSVFile(directoryString+"/ReportAONMaxSkillShortage_" + description + ".csv"));
         TempAverageSkillShortageResults.add(new
CSVFile(directoryString+"/ReportAONAverageSkillShortage_" + description +
".csv"));
         TempAliveAgentsResults.add(new
CSVFile(directoryString+"/ReportAONAliveAgents_" + description + ".csv"));

         TempSkillCountResults.add(new
CSVFile(directoryString+"/ReportAONSkillCount_" + description + ".csv"));
         TempSkillFailureCountResults.add(new
CSVFile(directoryString+"/ReportAONSkillFailureCount_" + description +
".csv"));
         TempSkillDemandCountResults.add(new
CSVFile(directoryString+"/ReportAONSkillDemandCount_" + description + ".csv"));

         Vector testAgentTypeCounts = new Vector();
         for(int j = 0; j < getLength(m_agentTypes); j++){
            String agentType = (String)getItem(j,m_agentTypes);
            testAgentTypeCounts.add(new CSVFile(directoryString+"/ReportAON" +
agentType + "Count_" + description + ".csv"));
         }//for
         TempAgentTypeCountResults.add(testAgentTypeCounts);
         headerValues.add(testName);
      }//for

      //Summarize Optimal Performance data for each adaptation method
      processOptimalPerformanceAll(headerValues,
TempOptimalPerformanceResults);
      //Summarize Performance data for each adaptation method
      processPerformanceAll(headerValues, TempPerformanceResults);
      //Summarize Efficiency for each adaptation method
      processEfficiencyAll(headerValues, TempEfficiencyResults);
      //Summarize Minimum Degree for each adaptation method
      processMinDegree(headerValues, TempMinDegreeResults);
      //Summarize Maximum Degree for each adaptation method
      processMaxDegree(headerValues, TempMaxDegreeResults);
      //Summarize Degree Variance for each adaptation method
      processDegreeVariance(headerValues, TempDegreeVarianceResults);
      //Summarize Average Degree for each adaptation method
      processAverageDegree(headerValues, TempAverageDegreeResults);
      //Summarize Average Density for each adaptation method
      processAverageDensity(headerValues, TempAverageDensityResults);
      //Summarize Changed Edges for each adaptation method
      processChangedEdges(headerValues, TempChangedEdgesResults);

      //Summarize Time Spent for each adaptation method
      processTimeSpent(headerValues, TempTimeWaitingResults,
TempTimeRewiringResults, TempTimeProposingResults, TempTimeJoiningResults,
TempTimeWorkingResults);

      processChangedAgents(headerValues, TempChangedAgentsResults);
      processTaskDistance(headerValues, TempTaskDistanceResults);

      processPendingFailure(headerValues, TempPendingFailureResults);
      processSkillFailure(headerValues, TempSkillFailureResults);
      processDistanceFailure(headerValues, TempDistanceFailureResults);
      processTaskComplete(headerValues, TempTaskCompleteResults);
```

```
      processTaskOutcome(headerValues, TempPendingFailureResults,
TempSkillFailureResults, TempDistanceFailureResults, TempTaskCompleteResults);

      processTaskClusters(headerValues, TempTaskClustersResults);
      processTaskCount(headerValues, TempTaskCountResults);
      processDistanceTravelled(headerValues, TempDistanceTravelledResults);

      processJoinedImpossible(headerValues, TempJoinedImpossibleResults);
      processProposedImpossible(headerValues, TempProposedImpossibleResults);
      processRepeatTask(headerValues, TempRepeatTaskResults);
      processMinSkillShortage(headerValues, TempMinSkillShortageResults);
      processMaxSkillShortage(headerValues, TempMaxSkillShortageResults);
      processAverageSkillShortage(headerValues,
TempAverageSkillShortageResults);
      processAliveAgents(headerValues, TempAliveAgentsResults);

      processSkillCount(headerValues, TempSkillCountResults);
      processSkillFailureCount(headerValues, TempSkillFailureCountResults);
      processSkillDemandCount(headerValues, TempSkillDemandCountResults);

      processInitialAgentTypeCount(headerValues, TempAgentTypeCountResults);
      processAgentTypeCount(headerValues, TempAgentTypeCountResults);

      for(int i = 0; i < m_numberOfTests; i++){
         String testName = (String)getItem(i,m_testNames);
         CSVFile AgentTypeCountResults = new CSVFile();
          Vector values = new Vector();
          values.add(" ");
          for(int j = 0; j < getLength(m_agentTypes); j++){
             String agentType = (String)getItem(j,m_agentTypes);
             values.add(testName+" "+agentType);
          }//for
          AgentTypeCountResults.addRow(values);

          Vector testAgentTypeCounts =
(Vector)getItem(i,TempAgentTypeCountResults);
          CSVFile FirstAgentTypeCountResults =
(CSVFile)getItem(0,testAgentTypeCounts);

          for(int j = 0; j < FirstAgentTypeCountResults.getLineCount(); j++){
             String timeStep = FirstAgentTypeCountResults.getItem(j,0);
             values = new Vector();
             values.add(timeStep);
             for(int k = 0; k < getLength(m_agentTypes); k++){

values.add(((CSVFile)getItem(k,testAgentTypeCounts)).getItem(j,m_averageColumnI
ndex));
              }//for
              for(int k = 0; k < getLength(m_agentTypes); k++){

values.add(((CSVFile)getItem(k,testAgentTypeCounts)).getItem(j,m_standardErrorC
olumnIndex));
              }//for
              AgentTypeCountResults.addRow(values);
          }//for

          String aonDirectoryString = getAONReportDirectory();
          AgentTypeCountResults.save(aonDirectoryString + "/" + testName +
"AgentCountResults" + m_outputDirectory + ".csv");

writePlotAgentTypeCountTimeSeriesFile(testName,AgentTypeCountResults);
       }//for
```

```
        writePlotAllFile();
        writePlotAllBatchFile();
   }//processReport

   private void getColumnIndices(CSVFile ResultsFile){
      int averageColumnIndex = -1;
      int standardErrorColumnIndex = -1;
      int improvementColumnIndex = -1;
      int row = 1, col = 0;
      String item = ResultsFile.getItem(row,col);
      while(item != null){
         System.out.print(item + " ");
         if(item.compareTo("average") == 0){
            averageColumnIndex = col;
         }else if(item.compareTo("2*StdErr") == 0){
            standardErrorColumnIndex = col;
         }else if(item.compareTo("improvement") == 0){
            improvementColumnIndex = col;
         }//else if
         item = ResultsFile.getItem(row,col);
         col++;
      }//while
      System.out.print("\n");
      if(averageColumnIndex-1 >= 0){
         m_averageColumnIndex = averageColumnIndex-1;
      }//if
      if(standardErrorColumnIndex-1 >= 0){
         m_standardErrorColumnIndex = standardErrorColumnIndex-1;
      }//if
      if(improvementColumnIndex-1 >= 0){
         m_improvementColumnIndex = improvementColumnIndex-1;
      }//if
   }//getColumnIndices

   private CSVFile getAverageResults(Vector headerValues, Vector TempResults,
String fileDescription){
      CSVFile AverageResults = new CSVFile();
      CSVFile FirstAverageResults = (CSVFile)getItem(0,TempResults);
      getColumnIndices(FirstAverageResults);
      System.out.println("Before loop: averageColumnIndex=" +
m_averageColumnIndex + " improvementColumnIndex=" + m_improvementColumnIndex);
      for(int i = 0; i < FirstAverageResults.getLineCount(); i++){
           Vector averageValues = new Vector();
           if(i == 0){
              AverageResults.addRow(headerValues);
           }else{
              String timeStep = FirstAverageResults.getItem(i,0);
              averageValues.add(timeStep);
              for(int j = 0; j < m_numberOfTests; j++){
                 CSVFile currentFile = (CSVFile)getItem(j,TempResults);

averageValues.add(currentFile.getItem(i,m_averageColumnIndex));
              }//for
              for(int j = 0; j < m_numberOfTests; j++){
                 CSVFile currentFile = (CSVFile)getItem(j,TempResults);

averageValues.add(currentFile.getItem(i,m_standardErrorColumnIndex));
              }//for
              AverageResults.addRow(averageValues);
          }//else
       }//for
       String aonDirectoryString = getAONReportDirectory();
```

```
       AverageResults.save(aonDirectoryString + "/" + fileDescription +
m_outputDirectory + ".csv");
       return AverageResults;
   }//getAverageResults

   private CSVFile getErrorResults(Vector headerValues, Vector TempResults,
String fileDescription){
      CSVFile ErrorResults = new CSVFile();
      CSVFile FirstErrorResults = (CSVFile)getItem(0,TempResults);
      getColumnIndices(FirstErrorResults);
      System.out.println("Before loop: averageColumnIndex=" +
m_averageColumnIndex + " improvementColumnIndex=" + m_improvementColumnIndex);
      for(int i = 0; i < FirstErrorResults.getLineCount(); i++){
           Vector errorValues = new Vector();
           if(i == 0){
              ErrorResults.addRow(headerValues);
           }else{
              String timeStep = FirstErrorResults.getItem(i,0);
              errorValues.add(timeStep);
              for(int j = 0; j < m_numberOfTests; j++){
                 CSVFile currentFile = (CSVFile)getItem(j,TempResults);

errorValues.add(currentFile.getItem(i,m_standardErrorColumnIndex));
              }//for
              ErrorResults.addRow(errorValues);
          }//else
       }//for
       String aonDirectoryString = getAONReportDirectory();
       ErrorResults.save(aonDirectoryString + "/" + fileDescription +
m_outputDirectory + ".csv");
       return ErrorResults;
   }//getErrorResults

   private CSVFile getImprovementResults(Vector headerValues, Vector
TempResults, String fileDescription){
      CSVFile ImprovementResults = new CSVFile();
      CSVFile FirstImprovementResults = (CSVFile)getItem(0,TempResults);
      getColumnIndices(FirstImprovementResults);
      System.out.println("Before loop: averageColumnIndex=" +
m_averageColumnIndex + " improvementColumnIndex=" + m_improvementColumnIndex);
      for(int i = 0; i < FirstImprovementResults.getLineCount(); i++){
           Vector improvementValues = new Vector();
           if(i == 0){
              ImprovementResults.addRow(headerValues);
           }else{
              String timeStep = FirstImprovementResults.getItem(i,0);
              improvementValues.add(timeStep);
              for(int j = 0; j < m_numberOfTests; j++){
                 CSVFile currentFile = (CSVFile)getItem(j,TempResults);

improvementValues.add(currentFile.getItem(i,m_improvementColumnIndex));
              }//for
              ImprovementResults.addRow(improvementValues);
          }//else
       }//for
       String aonDirectoryString = getAONReportDirectory();
       ImprovementResults.save(aonDirectoryString + "/" + fileDescription +
m_outputDirectory + ".csv");
       return ImprovementResults;
   }//getImprovementResults

   private void processOptimalPerformanceAll(Vector headerValues, Vector
TempOptimalPerformanceResults){
```

```
      String aonDirectoryString = getAONReportDirectory();
      CSVFile OptimalPerformanceResults =
getAverageResults(headerValues,TempOptimalPerformanceResults,"OptimalPerformanc
eResults");
      writePlotOptimalPerformanceFile(OptimalPerformanceResults);
   }//processOptimalPerformanceAll

   private void processPerformanceAll(Vector headerValues, Vector
TempPerformanceResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile PerformanceResults =
getAverageResults(headerValues,TempPerformanceResults,"PerformanceResults");
      CSVFile PerformanceErrorResults =
getErrorResults(headerValues,TempPerformanceResults,"PerformanceErrorResults");
      CSVFile PerformanceImprovementResults =
getImprovementResults(headerValues,TempPerformanceResults,"PerformanceImproveme
ntResults");
      writePlotPerformanceFile(PerformanceResults);
      writePlotPerformanceYErrorFile(PerformanceResults);
      writePlotPerformanceErrorFile(PerformanceErrorResults);
      writePlotPerformanceImprovementFile(PerformanceImprovementResults);
   }//processPerformanceAll

   private void processEfficiencyAll(Vector headerValues, Vector
TempEfficiencyResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile EfficiencyResults =
getAverageResults(headerValues,TempEfficiencyResults,"EfficiencyResults");
      CSVFile EfficiencyErrorResults =
getErrorResults(headerValues,TempEfficiencyResults,"EfficiencyErrorResults");
      writePlotEfficiencyFile(EfficiencyResults);
      writePlotEfficiencyYErrorFile(EfficiencyResults);
      writePlotEfficiencyErrorFile(EfficiencyErrorResults);
   }//processEfficiencyAll

   private void processMinDegree(Vector headerValues, Vector
TempMinDegreeResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile MinDegreeResults =
getAverageResults(headerValues,TempMinDegreeResults,"MinDegreeResults");
      writePlotMinDegreeFile(MinDegreeResults);
   }//processMinDegree

   private void processMaxDegree(Vector headerValues, Vector
TempMaxDegreeResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile MaxDegreeResults =
getAverageResults(headerValues,TempMaxDegreeResults,"MaxDegreeResults");
      writePlotMaxDegreeFile(MaxDegreeResults);
   }//processMaxDegree

   private void processDegreeVariance(Vector headerValues, Vector
TempDegreeVarianceResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile DegreeVarianceResults =
getAverageResults(headerValues,TempDegreeVarianceResults,"DegreeVarianceResults
");
      writePlotDegreeVarianceFile(DegreeVarianceResults);
   }//processDegreeVariance

   private void processAverageDegree(Vector headerValues, Vector
TempAverageDegreeResults){
      String aonDirectoryString = getAONReportDirectory();
```

```
      CSVFile AverageDegreeResults =
getAverageResults(headerValues,TempAverageDegreeResults,"AverageDegreeResults")
;
      writePlotAverageDegreeFile(AverageDegreeResults);
   }//processAverageDegree

   private void processAverageDensity(Vector headerValues, Vector
TempAverageDensityResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile AverageDensityResults =
getAverageResults(headerValues,TempAverageDensityResults,"AverageDensityResults
");
      writePlotAverageDensityFile(AverageDensityResults);
   }//processAverageDensity

   private void processChangedEdges(Vector headerValues, Vector
TempChangedEdgesResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile ChangedEdgesResults =
getAverageResults(headerValues,TempChangedEdgesResults,"ChangedEdgesResults");
      writePlotChangedEdgesFile(ChangedEdgesResults);
   }//processChangedEdges

   private void processTimeSpent(Vector headerValues, Vector
TempTimeWaitingResults, Vector TempTimeRewiringResults,
                                 Vector TempTimeProposingResults, Vector
TempTimeJoiningResults, Vector TempTimeWorkingResults){
      CSVFile TimeSpentResults = new CSVFile();
      CSVFile FirstTimeWaitingResults =
(CSVFile)getItem(0,TempTimeWaitingResults);
      int lastRowIndex = FirstTimeWaitingResults.getLineCount() - 1;
      Vector values = new Vector();
      values.add(" ");
      values.add("time_waiting");
      values.add("time_rewiring");
      values.add("time_proposing");
      values.add("time_joining");
      values.add("time_working");
      TimeSpentResults.addRow(values);

      for(int i = 0; i < m_numberOfTests; i++){
         values = new Vector();
         values.add((String)getItem(i,m_testNames));

values.add(((CSVFile)getItem(i,TempTimeWaitingResults)).getItem(lastRowIndex,m_
averageColumnIndex));

values.add(((CSVFile)getItem(i,TempTimeRewiringResults)).getItem(lastRowIndex,m
_averageColumnIndex));

values.add(((CSVFile)getItem(i,TempTimeProposingResults)).getItem(lastRowIndex,
m_averageColumnIndex));

values.add(((CSVFile)getItem(i,TempTimeJoiningResults)).getItem(lastRowIndex,m_
averageColumnIndex));

values.add(((CSVFile)getItem(i,TempTimeWorkingResults)).getItem(lastRowIndex,m_
averageColumnIndex));
         TimeSpentResults.addRow(values);
      }//for

      String aonDirectoryString = getAONReportDirectory();
```

```
      TimeSpentResults.save(aonDirectoryString + "/TimeSpentResults" +
m_outputDirectory + ".csv");
      writeTimeSpentFile(TimeSpentResults);
      for(int i = 1; i <= m_numberOfTests; i++){
         writeTimeSpentPieChart(i,TimeSpentResults);
      }//for
   }//processTimeSpent

   private void processChangedAgents(Vector headerValues, Vector
TempChangedAgentsResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile ChangedAgentsResults =
getAverageResults(headerValues,TempChangedAgentsResults,"ChangedAgentsResults")
;
      writePlotChangedAgentsFile(ChangedAgentsResults);
   }//processChangedAgents

   private void processTaskDistance(Vector headerValues, Vector
TempTaskDistanceResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile TaskDistanceResults =
getAverageResults(headerValues,TempTaskDistanceResults,"TaskDistanceResults");
      writePlotTaskDistanceFile(TaskDistanceResults);
   }//processTaskDistance

   private void processPendingFailure(Vector headerValues, Vector
TempPendingFailureResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile PendingFailureResults =
getAverageResults(headerValues,TempPendingFailureResults,"PendingFailureResults
");
      writePlotPendingFailureFile(PendingFailureResults);
   }//processPendingFailure

   private void processSkillFailure(Vector headerValues, Vector
TempSkillFailureResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile SkillFailureResults =
getAverageResults(headerValues,TempSkillFailureResults,"SkillFailureResults");
      writePlotSkillFailureFile(SkillFailureResults);
   }//processSkillFailure

   private void processDistanceFailure(Vector headerValues, Vector
TempDistanceFailureResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile DistanceFailureResults =
getAverageResults(headerValues,TempDistanceFailureResults,"DistanceFailureResul
ts");
      writePlotDistanceFailureFile(DistanceFailureResults);
   }//processDistanceFailure

   private void processTaskComplete(Vector headerValues, Vector
TempTaskCompleteResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile TaskCompleteResults =
getAverageResults(headerValues,TempTaskCompleteResults,"TaskCompleteResults");
      writePlotTaskCompleteFile(TaskCompleteResults);
   }//processTaskComplete

   private void processTaskOutcome(Vector headerValues, Vector
TempPendingFailureResults, Vector TempSkillFailureResults,
                                   Vector TempDistanceFailureResults, Vector
TempTaskCompleteResults){
```

```
      CSVFile TaskOutcomeResults = new CSVFile();
      CSVFile FirstPendingFailureResults =
(CSVFile)getItem(0,TempPendingFailureResults);
      int lastRowIndex = FirstPendingFailureResults.getLineCount() - 1;
      Vector values = new Vector();
      values.add(" ");
      values.add("pending_failure");
      values.add("skill_failure");
      values.add("distance_failure");
      values.add("task_complete");
      TaskOutcomeResults.addRow(values);

      for(int i = 0; i < m_numberOfTests; i++){
         values = new Vector();
         values.add((String)getItem(i,m_testNames));

values.add(((CSVFile)getItem(i,TempPendingFailureResults)).getItem(lastRowIndex
,m_averageColumnIndex));

values.add(((CSVFile)getItem(i,TempSkillFailureResults)).getItem(lastRowIndex,m
_averageColumnIndex));

values.add(((CSVFile)getItem(i,TempDistanceFailureResults)).getItem(lastRowInde
x,m_averageColumnIndex));

values.add(((CSVFile)getItem(i,TempTaskCompleteResults)).getItem(lastRowIndex,m
_averageColumnIndex));
         TaskOutcomeResults.addRow(values);
      }//for

      String aonDirectoryString = getAONReportDirectory();
      TaskOutcomeResults.save(aonDirectoryString + "/TaskOutcomeResults" +
m_outputDirectory + ".csv");
      writeTaskOutcomeFile(TaskOutcomeResults);
      writeTaskOutcomePieChart(TaskOutcomeResults);
   }//processTaskOutcome

   private void processTaskClusters(Vector headerValues, Vector
TempTaskClustersResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile TaskClustersResults =
getAverageResults(headerValues,TempTaskClustersResults,"TaskClustersResults");
      writePlotTaskClustersFile(TaskClustersResults);
   }//processTaskClusters

   private void processTaskCount(Vector headerValues, Vector
TempTaskCountResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile TaskCountResults =
getAverageResults(headerValues,TempTaskCountResults,"TaskCountResults");
      writePlotTaskCountFile(TaskCountResults);
   }//processTaskCount

   private void processDistanceTravelled(Vector headerValues, Vector
TempDistanceTravelledResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile DistanceTravelledResults =
getAverageResults(headerValues,TempDistanceTravelledResults,"DistanceTravelledR
esults");
      writePlotDistanceTravelledFile(DistanceTravelledResults);
   }//processDistanceTravelled
```

```
   private void processJoinedImpossible(Vector headerValues, Vector
TempJoinedImpossibleResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile JoinedImpossibleResults =
getAverageResults(headerValues,TempJoinedImpossibleResults,"JoinedImpossibleRes
ults");
      writePlotJoinedImpossibleFile(JoinedImpossibleResults);
   }//processJoinedImpossible

   private void processProposedImpossible(Vector headerValues, Vector
TempProposedImpossibleResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile ProposedImpossibleResults =
getAverageResults(headerValues,TempProposedImpossibleResults,"ProposedImpossibl
eResults");
      writePlotProposedImpossibleFile(ProposedImpossibleResults);
   }//processProposedImpossible

   private void processRepeatTask(Vector headerValues, Vector
TempRepeatTaskResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile RepeatTaskResults =
getAverageResults(headerValues,TempRepeatTaskResults,"RepeatTaskResults");
      writePlotRepeatTaskFile(RepeatTaskResults);
   }//processRepeatTask

   private void processMinSkillShortage(Vector headerValues, Vector
TempMinSkillShortageResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile MinSkillShortageResults =
getAverageResults(headerValues,TempMinSkillShortageResults,"MinSkillShortageRes
ults");
      writePlotMinSkillShortageFile(MinSkillShortageResults);
   }//processMinSkillShortage

   private void processMaxSkillShortage(Vector headerValues, Vector
TempMaxSkillShortageResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile MaxSkillShortageResults =
getAverageResults(headerValues,TempMaxSkillShortageResults,"MaxSkillShortageRes
ults");
      writePlotMaxSkillShortageFile(MaxSkillShortageResults);
   }//processMaxSkillShortage

   private void processAverageSkillShortage(Vector headerValues, Vector
TempAverageSkillShortageResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile AverageSkillShortageResults =
getAverageResults(headerValues,TempAverageSkillShortageResults,"AverageSkillSho
rtageResults");
      writePlotAverageSkillShortageFile(AverageSkillShortageResults);
   }//processAverageSkillShortage

   private void processAliveAgents(Vector headerValues, Vector
TempAliveAgentsResults){
      String aonDirectoryString = getAONReportDirectory();
      CSVFile AliveAgentsResults =
getAverageResults(headerValues,TempAliveAgentsResults,"AliveAgentsResults");
      writePlotAliveAgentsFile(AliveAgentsResults);
   }//processAliveAgents

   private void processSkillCount(Vector headerValues, Vector
TempSkillCountResults){
```

```
      CSVFile SkillCountResults = new CSVFile();
      CSVFile FirstSkillCountResults =
(CSVFile)getItem(0,TempSkillCountResults);
      Vector values = new Vector();
      values.add(" ");
      for(int i = 1; i < FirstSkillCountResults.getLineCount(); i++){
         String skillName = FirstSkillCountResults.getItem(i,0);
         values.add(skillName);
      }//for
      SkillCountResults.addRow(values);

      for(int i = 0; i < m_numberOfTests; i++){
         String testName = (String)getItem(i,m_testNames);
         CSVFile CurrentSkillCountResults =
(CSVFile)getItem(i,TempSkillCountResults);
         values = new Vector();

         if(testName != null && testName != ""){
            values.add(testName);
         }else{
            values.add(" ");
         }//else

         for(int j = 1; j < CurrentSkillCountResults.getLineCount(); j++){
            String skillCount = CurrentSkillCountResults.getItem(j,1);
            values.add(skillCount);
         }//for
         SkillCountResults.addRow(values);
      }//for

      String aonDirectoryString = getAONReportDirectory();
      SkillCountResults.save(aonDirectoryString + "/SkillCountResults" +
m_outputDirectory + ".csv");
      writePlotSkillCount(SkillCountResults);
      //writeAgentCountPieChart(SkillCountResults);
   }//processSkillCount

   private void processSkillFailureCount(Vector headerValues, Vector
TempSkillFailureCountResults){
      CSVFile SkillFailureCountResults = new CSVFile();
      CSVFile FirstSkillFailureCountResults =
(CSVFile)getItem(0,TempSkillFailureCountResults);
      Vector values = new Vector();
      values.add(" ");
      for(int i = 1; i < FirstSkillFailureCountResults.getLineCount(); i++){
         String skillName = FirstSkillFailureCountResults.getItem(i,0);
         values.add(skillName);
         System.out.println("Skill Name: " + skillName);
      }//for
      SkillFailureCountResults.addRow(values);

      for(int i = 0; i < m_numberOfTests; i++){
         String testName = (String)getItem(i,m_testNames);
         CSVFile CurrentSkillFailureCountResults =
(CSVFile)getItem(i,TempSkillFailureCountResults);
         values = new Vector();

         if(testName != null && testName != ""){
            values.add(testName);
         }else{
            values.add(" ");
         }//else
```

```
         for(int j = 1; j < CurrentSkillFailureCountResults.getLineCount();
j++){
            String skillCount = CurrentSkillFailureCountResults.getItem(j,1);
            values.add(skillCount);
         }//for
         SkillFailureCountResults.addRow(values);
      }//for

      String aonDirectoryString = getAONReportDirectory();
      SkillFailureCountResults.save(aonDirectoryString +
"/SkillFailureCountResults" + m_outputDirectory + ".csv");
      writePlotSkillFailureCount(SkillFailureCountResults);
      //writeAgentCountPieChart(SkillFailureCountResults);
   }//processSkillFailureCount

   private void processSkillDemandCount(Vector headerValues, Vector
TempSkillDemandCountResults){
      CSVFile SkillDemandCountResults = new CSVFile();
      CSVFile FirstSkillDemandCountResults =
(CSVFile)getItem(0,TempSkillDemandCountResults);
      Vector values = new Vector();
      values.add(" ");
      for(int i = 1; i < FirstSkillDemandCountResults.getLineCount(); i++){
         String skillName = FirstSkillDemandCountResults.getItem(i,0);
         values.add(skillName);
         System.out.println("Skill Name: " + skillName);
      }//for
      SkillDemandCountResults.addRow(values);

      for(int i = 0; i < m_numberOfTests; i++){
         String testName = (String)getItem(i,m_testNames);
         CSVFile CurrentSkillDemandCountResults =
(CSVFile)getItem(i,TempSkillDemandCountResults);
         values = new Vector();

         if(testName != null && testName != ""){
            values.add(testName);
         }else{
            values.add(" ");
         }//else

         for(int j = 1; j < CurrentSkillDemandCountResults.getLineCount();
j++){
            String skillCount = CurrentSkillDemandCountResults.getItem(j,1);
            values.add(skillCount);
         }//for
         SkillDemandCountResults.addRow(values);
      }//for

      String aonDirectoryString = getAONReportDirectory();
      SkillDemandCountResults.save(aonDirectoryString +
"/SkillDemandCountResults" + m_outputDirectory + ".csv");
      writePlotSkillDemandCount(SkillDemandCountResults);
      //writeAgentCountPieChart(SkillDemandCountResults);
   }//processSkillDemandCount

   private void processInitialAgentTypeCount(Vector headerValues, Vector
TempAgentTypeCountResults){
      CSVFile AgentTypeCountResults = new CSVFile();
      Vector values = new Vector();
      values.add(" ");
      for(int i = 0; i < getLength(m_agentTypes); i++){
         String agentType = (String)getItem(i,m_agentTypes);
```

```
         values.add(agentType);
      }//for
      AgentTypeCountResults.addRow(values);

      for(int i = 0; i < m_numberOfTests; i++){
         String testName = (String)getItem(i,m_testNames);
         Vector testAgentTypeCounts =
(Vector)getItem(i,TempAgentTypeCountResults);
         values = new Vector();
         if(testName != null && testName != ""){
            values.add(testName);
         }else{
            values.add(" ");
         }//else
         for(int j = 0; j < getLength(m_agentTypes); j++){
            String agentType = (String)getItem(j,m_agentTypes);

values.add(((CSVFile)getItem(j,testAgentTypeCounts)).getItem(2,m_averageColumnI
ndex));
         }//for
         AgentTypeCountResults.addRow(values);
      }//for

      String aonDirectoryString = getAONReportDirectory();
      AgentTypeCountResults.save(aonDirectoryString +
"/InitialAgentCountResults" + m_outputDirectory + ".csv");
      writeInitialAgentCountFile(AgentTypeCountResults);
   }//processInitialAgentTypeCount

   private void processAgentTypeCount(Vector headerValues, Vector
TempAgentTypeCountResults){
      CSVFile AgentTypeCountResults = new CSVFile();
      Vector values = new Vector();
      values.add(" ");
      for(int i = 0; i < getLength(m_agentTypes); i++){
         String agentType = (String)getItem(i,m_agentTypes);
         values.add(agentType);
      }//for
      AgentTypeCountResults.addRow(values);

      for(int i = 0; i < m_numberOfTests; i++){
         String testName = (String)getItem(i,m_testNames);
         Vector testAgentTypeCounts =
(Vector)getItem(i,TempAgentTypeCountResults);
         CSVFile FirstAgentTypeCountResults =
(CSVFile)getItem(0,testAgentTypeCounts);
         int lastRowIndex = FirstAgentTypeCountResults.getLineCount() - 1;
         values = new Vector();

         if(testName != null && testName != ""){
            values.add(testName);
         }else{
            values.add(" ");
         }//else

         for(int j = 0; j < getLength(m_agentTypes); j++){
            String agentType = (String)getItem(j,m_agentTypes);

values.add(((CSVFile)getItem(j,testAgentTypeCounts)).getItem(lastRowIndex,m_ave
rageColumnIndex));
         }//for
         AgentTypeCountResults.addRow(values);
      }//for
```

```
      String aonDirectoryString = getAONReportDirectory();
      AgentTypeCountResults.save(aonDirectoryString + "/AgentCountResults" +
m_outputDirectory + ".csv");
      writeAgentCountFile(AgentTypeCountResults);
      writeAgentCountPieChart(AgentTypeCountResults);
   }//processAgentTypeCount

   private String getAONReportDirectory(){
      String directoryString = "reports";
      File reportDirectory = new File(directoryString);
      if(!reportDirectory.exists()){
         reportDirectory.mkdir();
      }//if

      String aonDirectoryString = "reports/" + m_outputDirectory;
      File aonReportDirectory = new File(aonDirectoryString);
      if(!aonReportDirectory.exists()){
         aonReportDirectory.mkdir();
      }//if

      return aonDirectoryString;
   }//getAONReportDirectory

   private void readInputFile(){
      System.out.println("readInputFile");
      AONXMLReader xmlFile = new AONXMLReader();
      if(!xmlFile.openXML(m_inputFilename)){
         System.out.println("Bad XML File: " + m_inputFilename + " Using
default AON settings");
         return;
      }//if

      XMLElement root = xmlFile.getRoot();
      String attributeName = "";
      String value = "";

      String testName = "";
      String outputDirectory = "";
      String description = "";

      try{
         attributeName = "ReportDirectory";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_outputDirectory = value;

         attributeName = "ReportDescription";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_reportDescription = value;

         attributeName = "MinAdaptTimeStep";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_minAdaptTimeStep =
Integer.parseInt(value);

         attributeName = "NumberOfSkills";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_numberOfSkills =
Integer.parseInt(value);
      }catch(Exception ex){
      }//catch

      try{
```

```
         XMLElement xmlAgentTypes = root.getElement("AgentTypes");
         if(xmlAgentTypes != null){
            int agentTypesCount = xmlAgentTypes.getNumElements();
            for(int i = 0; i < agentTypesCount; i++){
               XMLElement xmlAgentType = xmlAgentTypes.getElement(i);
               String type = xmlAgentType.getAttribute("Name");
               String colorName = xmlAgentType.getAttribute("Color");
               m_agentTypes.add(type);
            }//for
         }//if
         System.out.println("Read Agent Types from " + m_inputFilename);
      }catch(Exception ex){
         System.out.println("Error reading Agent Types from xml file: " + ex);
      }//catch

      XMLElement xmlTests = root.getElement("Tests");
      m_numberOfTests = xmlTests.getNumElements();
      for(int i = 0; i < m_numberOfTests; i++){
         XMLElement xmlTest = xmlTests.getElement(i);
         try{
            attributeName = "Name";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)testName = value;

            attributeName = "OutputDirectory";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)outputDirectory = value;

            attributeName = "Description";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)description = value;

            System.out.println("Read AON parameters from " + m_inputFilename);
         }catch(Exception ex){
            System.out.println("Error loading test attribute " + attributeName
+ "from xml file: " + ex.getMessage());
         }//catch
         System.out.println("Name=" + testName + " directory=" +
outputDirectory + " description=" + description);
         m_testNames.add(testName);
         m_outputDirectories.add(outputDirectory);
         m_descriptions.add(description);
      }//for
   }//readInputFile

   private int getMaxTimeStep(CSVFile file){
      int maxTimeStep = file.getLineCount()-1;
      try{
         String value = file.getItem(file.getLineCount()-1,0);
         int temp = (int)Float.parseFloat(value);
         maxTimeStep = temp;
      }catch(Exception ex){
      }//catch
      return maxTimeStep;
   }//getMaxTimeStep

   private void writePlotAllFile(){
      String filename = getAONReportDirectory() + "/graph_all.plt";
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(filename));
         for(int i = 0; i < getLength(m_plotFileNames); i++){
            String plotFilename = (String)getItem(i,m_plotFileNames);
            out.write("load \"" + plotFilename + "\"");out.newLine();
```

```
         }//for
         out.close();
         System.out.println("Wrote file: " + filename);
      }catch(IOException ex){
         System.out.println("Error writing file " + filename + ": " +
ex.getMessage());
      }//catch
   }//writePlotAllFile

   private void writePlotAllBatchFile(){
      String filename = getAONReportDirectory() + "/graph_all.bat";
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(filename));
         out.write("\"C:\\Program Files\\gnuplot\\wgnuplot.exe\"
graph_all.plt");out.newLine();
         out.close();
         System.out.println("Wrote file: " + filename);
      }catch(IOException ex){
         System.out.println("Error writing file " + filename + ": " +
ex.getMessage());
      }//catch
   }//writePlotAllBatchFile

   private void writePlotOptimalPerformanceFile(CSVFile
OptimalPerformanceResults){
      String plotName = "graph_optimal_performance.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Optimal Performance");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("optimal performance");

gnuplot.plotLinesPoints(OptimalPerformanceResults,m_testNames,"OptimalPerforman
ceResults","OptimalPerformance",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotOptimalPerformanceFile

   private void writePlotPerformanceFile(CSVFile PerformanceResults){
      String plotName = "graph_performance.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Performance");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("performance");

gnuplot.plotLinesPoints(PerformanceResults,m_testNames,"PerformanceResults","Pe
rformance",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotPerformanceFile

   private void writePlotPerformanceYErrorFile(CSVFile PerformanceResults){
      String plotName = "graph_performance_yerror.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Performance");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("performance");
```

```
gnuplot.plotYErrorBars(PerformanceResults,m_testNames,"PerformanceResults","Per
formanceYErrors",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotPerformanceYErrorFile

   private void writePlotPerformanceErrorFile(CSVFile PerformanceErrorResults){
      String plotName = "graph_performance_error.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Performance Error");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("performance error");

gnuplot.plotLinesPoints(PerformanceErrorResults,m_testNames,"PerformanceErrorRe
sults","PerformanceError",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotPerformanceErrorFile

   private void writePlotPerformanceImprovementFile(CSVFile
PerformanceImprovementResults){
      String plotName = "graph_performance_improvement.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Performance Improvement");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("performance improvement");

gnuplot.plotLinesPoints(PerformanceImprovementResults,m_testNames,"PerformanceI
mprovementResults","PerformanceImprovement",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotPerformanceImprovementFile

   private void writePlotEfficiencyFile(CSVFile EfficiencyResults){
      String plotName = "graph_efficiency.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Efficiency");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("efficiency");

gnuplot.plotLinesPoints(EfficiencyResults,m_testNames,"EfficiencyResults","Effi
ciency",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotEfficiencyFile

   private void writePlotEfficiencyYErrorFile(CSVFile EfficiencyResults){
      String plotName = "graph_efficiency_yerror.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Efficiency");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("efficiency");
```

```
gnuplot.plotYErrorBars(EfficiencyResults,m_testNames,"EfficiencyResults","Effic
iencyYErrors",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotEfficiencyYErrorFile

   private void writePlotEfficiencyErrorFile(CSVFile EfficiencyErrorResults){
      String plotName = "graph_efficiency_error.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Efficiency Error");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("efficiency error");

gnuplot.plotLinesPoints(EfficiencyErrorResults,m_testNames,"EfficiencyErrorResu
lts","EfficiencyError",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotEfficiencyErrorFile

   private void writePlotMinDegreeFile(CSVFile MinDegreeResults){
      String plotName = "graph_min_degree.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Minimum Degree");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("minimum degree");

gnuplot.plotLinesPoints(MinDegreeResults,m_testNames,"MinDegreeResults","MinDeg
ree",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotMinDegreeFile

   private void writePlotMaxDegreeFile(CSVFile MaxDegreeResults){
      String plotName = "graph_max_degree.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Maximum Degree");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("maximum degree");

gnuplot.plotLinesPoints(MaxDegreeResults,m_testNames,"MaxDegreeResults","MaxDeg
ree",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotMaxDegreeFile

   private void writePlotDegreeVarianceFile(CSVFile DegreeVarianceResults){
      String plotName = "graph_degree_variance.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Degree Variance");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("degree variance");

gnuplot.plotLinesPoints(DegreeVarianceResults,m_testNames,"DegreeVarianceResult
s","DegreeVariance",m_outputDirectory);
```

```
      gnuplot.save(filename);
   }//writePlotDegreeVarianceFile

   private void writePlotAverageDegreeFile(CSVFile AverageDegreeResults){
      String plotName = "graph_average_degree.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Average Degree");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("average degree");

gnuplot.plotLinesPoints(AverageDegreeResults,m_testNames,"AverageDegreeResults"
,"AverageDegree",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotAverageDegreeFile

   private void writePlotAverageDensityFile(CSVFile AverageDensityResults){
      String plotName = "graph_average_density.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Average Density");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("average density");

gnuplot.plotLinesPoints(AverageDensityResults,m_testNames,"AverageDensityResult
s","AverageDensity",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotAverageDensityFile

   private void writePlotChangedEdgesFile(CSVFile ChangedEdgesResults){
      String plotName = "graph_changed_edges.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Changed Edges");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("changed edges");

gnuplot.plotLinesPoints(ChangedEdgesResults,m_testNames,"ChangedEdgesResults","
ChangedEdges",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotChangedEdgesFile

   private void writeTimeSpentFile(CSVFile TimeSpentResults){
      int max_x = getMaxTimeStep(TimeSpentResults);
      int min_x = 0;
      float max_y = TimeSpentResults.getMaxValue() +
TimeSpentResults.getMaxValue()*0.3f;
      if(max_y == 0.0f){
         max_y = 0.1f;
      }//if
      int min_y = 0;
      String filename = getAONReportDirectory() + "/graph_time_spent.plt";
      m_plotFileNames.add("graph_time_spent.plt");
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(filename));
         out.write("set title \"AON Time Spent\\n " + m_reportDescription +
"\"");out.newLine();
```

```
         out.write("set style data histogram");out.newLine();
         out.write("set style histogram cluster gap 5");out.newLine();
         out.write("set style fill solid border -1");out.newLine();
         out.write("set boxwidth 0.9");out.newLine();
         out.write("set xtic rotate by -45");out.newLine();
         out.write("set bmargin 10");out.newLine();
         out.write("set yrange [ 0.00000 : 250000.0 ] noreverse
nowriteback");out.newLine();
         out.write("set datafile separator \",\"");out.newLine();
         out.write("plot 'TimeSpentResults" + m_outputDirectory + ".csv' using
2:xtic(1) title col, \\");out.newLine();
         out.write("   '' u 3 ti col, '' u 4 ti col, '' u 5 ti col, '' u 6 ti
col");out.newLine();
         out.write("set terminal emf  'Times Roman Bold' 18");out.newLine();
         out.write("set output \"" + m_outputDirectory +
"_TimeSpent.emf\"");out.newLine();
         out.write("replot");out.newLine();
         out.write("set output");out.newLine();
         out.write("set terminal windows");out.newLine();
         out.write("reset");out.newLine();
         out.close();
         System.out.println("Wrote file: " + filename);
      }catch(IOException ex){
         System.out.println("Error writing file " + filename + ": " +
ex.getMessage());
      }//catch*/
   }//writeTimeSpentFile

   private void writeTimeSpentPieChart(int agentValuesRow, CSVFile
TimeSpentResults){
      String agentName = TimeSpentResults.getItem(agentValuesRow,0);
      PieChart pie = new PieChart();
      pie.setTitle("AON Time Spent\n " + agentName);
      pie.setHeaderRow(0);
      pie.setValuesRow(agentValuesRow);
      pie.setStartColumn(1);

pie.createPieChart(getAONReportDirectory(),"AONTimeSpent"+agentName,TimeSpentRe
sults);
   }//writeTimeSpentPieChart

   private void writePlotChangedAgentsFile(CSVFile ChangedAgentsResults){
      String plotName = "graph_changed_agents.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Changed Agents");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("changed agents");

gnuplot.plotLinesPoints(ChangedAgentsResults,m_testNames,"ChangedAgentsResults"
,"ChangedAgents",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotChangedAgentsFile

   private void writePlotTaskDistanceFile(CSVFile TaskDistanceResults){
      String plotName = "graph_task_distance.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Average Task Distance");
      gnuplot.setSubtitle(m_reportDescription);
```

```
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("average task distance");

gnuplot.plotLinesPoints(TaskDistanceResults,m_testNames,"TaskDistanceResults","
TaskDistance",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotTaskDistanceFile

   private void writePlotPendingFailureFile(CSVFile PendingFailureResults){
      String plotName = "graph_pending_failures.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Pending Failures");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("pending failures");

gnuplot.plotLinesPoints(PendingFailureResults,m_testNames,"PendingFailureResult
s","PendingFailure",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotPendingFailureFile

   private void writePlotSkillFailureFile(CSVFile SkillFailureResults){
      String plotName = "graph_skill_failures.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Skill Failures");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("skill failures");

gnuplot.plotLinesPoints(SkillFailureResults,m_testNames,"SkillFailureResults","
SkillFailure",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotSkillFailureFile

   private void writePlotDistanceFailureFile(CSVFile DistanceFailureResults){
      String plotName = "graph_distance_failures.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Distance Failures");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("distance failures");

gnuplot.plotLinesPoints(DistanceFailureResults,m_testNames,"DistanceFailureResu
lts","DistanceFailure",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotDistanceFailureFile

   private void writePlotTaskCompleteFile(CSVFile TaskCompleteResults){
      String plotName = "graph_tasks_completed.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Tasks Completed");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("tasks completed");
```

```
gnuplot.plotLinesPoints(TaskCompleteResults,m_testNames,"TaskCompleteResults","
TaskComplete",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotTaskCompleteFile

   private void writeTaskOutcomeFile(CSVFile TaskOutcomeResults){
      int max_x = getMaxTimeStep(TaskOutcomeResults);
      int min_x = 0;
      float max_y = TaskOutcomeResults.getMaxValue() +
TaskOutcomeResults.getMaxValue()*0.3f;
      if(max_y == 0.0f){
         max_y = 0.1f;
      }//if
      int min_y = 0;
      String filename = getAONReportDirectory() + "/graph_task_outcome.plt";
      m_plotFileNames.add("graph_task_outcome.plt");
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(filename));
         out.write("set title \"AON Task Outcome\\n " + m_reportDescription +
"\"");out.newLine();
         out.write("set style data histogram");out.newLine();
         out.write("set style histogram cluster gap 5");out.newLine();
         out.write("set style fill solid border -1");out.newLine();
         out.write("set boxwidth 0.9");out.newLine();
         out.write("set xtic rotate by -45");out.newLine();
         out.write("set bmargin 10");out.newLine();
         out.write("set yrange [ 0.00000 : 250000.0 ] noreverse
nowriteback");out.newLine();
         out.write("set datafile separator \",\"");out.newLine();
         out.write("plot 'TaskOutcomeResults" + m_outputDirectory + ".csv'
using 2:xtic(1) title col, \\");out.newLine();
         out.write("   '' u 3 ti col, '' u 4 ti col, '' u 5 ti
col");out.newLine();
         out.write("set terminal emf  'Times Roman Bold' 18");out.newLine();
         out.write("set output \"" + m_outputDirectory +
"_TaskOutcome.emf\"");out.newLine();
         out.write("replot");out.newLine();
         out.write("set output");out.newLine();
         out.write("set terminal windows");out.newLine();
         out.write("reset");out.newLine();
         out.close();
         System.out.println("Wrote file: " + filename);
      }catch(IOException ex){
         System.out.println("Error writing file " + filename + ": " +
ex.getMessage());
      }//catch*/
   }//writeTaskOutcomeFile

   private void writeTaskOutcomePieChart(CSVFile TaskOutcomeResults){
      int agentValuesRow = 1;
      String agentName = TaskOutcomeResults.getItem(agentValuesRow,0);
      PieChart pie = new PieChart();
      pie.setTitle("AON Task Outcome\n " + agentName);
      pie.setHeaderRow(0);
      pie.setValuesRow(agentValuesRow);
      pie.setStartColumn(1);

pie.createPieChart(getAONReportDirectory(),"AONTaskOutcome"+agentName,TaskOutco
meResults);
   }//writeTaskOutcomePieChart

   private void writePlotTaskClustersFile(CSVFile TaskClustersResults){
```

```
      String plotName = "graph_task_clusters.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Task Clusters");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("task clusters");

gnuplot.plotLinesPoints(TaskClustersResults,m_testNames,"TaskClustersResults","
TaskClusters",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotTaskClustersFile

   private void writePlotTaskCountFile(CSVFile TaskCountResults){
      String plotName = "graph_task_count.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Task Count");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("task count");

gnuplot.plotLinesPoints(TaskCountResults,m_testNames,"TaskCountResults","TaskCo
unt",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotTaskCountFile

   private void writePlotDistanceTravelledFile(CSVFile
DistanceTravelledResults){
      String plotName = "graph_distance_travelled.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Distance Travelled");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("distance travelled");

gnuplot.plotLinesPoints(DistanceTravelledResults,m_testNames,"DistanceTravelled
Results","DistanceTravelled",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotDistanceTravelledFile

   private void writePlotJoinedImpossibleFile(CSVFile JoinedImpossibleResults){
      String plotName = "graph_joined_impossible.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Joined Impossible");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("joined impossible");

gnuplot.plotLinesPoints(JoinedImpossibleResults,m_testNames,"JoinedImpossibleRe
sults","JoinedImpossible",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotJoinedImpossibleFile

   private void writePlotProposedImpossibleFile(CSVFile
ProposedImpossibleResults){
      String plotName = "graph_proposed_impossible.plt";
```

```
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Proposed Impossible");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("proposed impossible");

gnuplot.plotLinesPoints(ProposedImpossibleResults,m_testNames,"ProposedImpossib
leResults","ProposedImpossible",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotProposedImpossibleFile

   private void writePlotRepeatTaskFile(CSVFile RepeatTaskResults){
      String plotName = "graph_repeat_task.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Repeat Task");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("repeat task");

gnuplot.plotLinesPoints(RepeatTaskResults,m_testNames,"RepeatTaskResults","Repe
atTask",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotRepeatTaskFile

   private void writePlotMinSkillShortageFile(CSVFile MinSkillShortageResults){
      String plotName = "graph_min_skill_shortage.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Minimum Skill Shortage");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("minimum skill shortage");

gnuplot.plotLinesPoints(MinSkillShortageResults,m_testNames,"MinSkillShortageRe
sults","MinSkillShortage",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotMinSkillShortageFile

   private void writePlotMaxSkillShortageFile(CSVFile MaxSkillShortageResults){
      String plotName = "graph_max_skill_shortage.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Maximum Skill Shortage");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("maximum skill shortage");

gnuplot.plotLinesPoints(MaxSkillShortageResults,m_testNames,"MaxSkillShortageRe
sults","MaxSkillShortage",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotMaxSkillShortageFile

   private void writePlotAverageSkillShortageFile(CSVFile
AverageSkillShortageResults){
      String plotName = "graph_average_skill_shortage.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
```

```
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Average Skill Shortage");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("average skill shortage");

gnuplot.plotLinesPoints(AverageSkillShortageResults,m_testNames,"AverageSkillSh
ortageResults","AverageSkillShortage",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotAverageSkillShortageFile

   private void writePlotAliveAgentsFile(CSVFile AliveAgentsResults){
      String plotName = "graph_alive_agents.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON Alive Agents");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("alive agents");

gnuplot.plotLinesPoints(AliveAgentsResults,m_testNames,"AliveAgentsResults","Al
iveAgents",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotAliveAgentsFile

   private void writePlotSkillCount(CSVFile SkillCountResults){
      int max_x = getMaxTimeStep(SkillCountResults);
      int min_x = 0;
      float max_y = SkillCountResults.getMaxValue() +
SkillCountResults.getMaxValue()*0.3f;
      if(max_y == 0.0f){
         max_y = 0.1f;
      }//if
      int min_y = 0;
      String filename = getAONReportDirectory() + "/graph_skill_count.plt";
      m_plotFileNames.add("graph_skill_count.plt");
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(filename));
         out.write("set title \"AON Skill Count\\n " + m_reportDescription +
"\"");out.newLine();
         out.write("set style data histogram");out.newLine();
         out.write("set style histogram cluster gap 5");out.newLine();
         out.write("set style fill solid border -1");out.newLine();
         out.write("set boxwidth 0.9");out.newLine();
         out.write("set xtic rotate by -45");out.newLine();
         out.write("set bmargin 10");out.newLine();
         out.write("set yrange [ 0.00000 : " + max_y + " ] noreverse
nowriteback");out.newLine();
         out.write("set datafile separator \",\"");out.newLine();
         if(m_numberOfSkills > 1){
            out.write("plot 'SkillCountResults" + m_outputDirectory + ".csv'
using 2:xtic(1) title col, \\");out.newLine();
         }else{
            out.write("plot 'SkillCountResults" + m_outputDirectory + ".csv'
using 2:xtic(1) title col");out.newLine();
         }//else
         for(int i = 3; i <= m_numberOfSkills+1; i++){
            if(i == 3){
               out.write("   '' u " + 3 + " ti col");
            }else{
               out.write(", '' u " + i + " ti col");
            }//else
```

```
         }//for
         out.newLine();
         out.write("set terminal emf  'Times Roman Bold' 18");out.newLine();
         out.write("set output \"" + m_outputDirectory +
"_SkillCount.emf\"");out.newLine();
         out.write("replot");out.newLine();
         out.write("set output");out.newLine();
         out.write("set terminal windows");out.newLine();
         out.write("reset");out.newLine();
         out.close();
         System.out.println("Wrote file: " + filename);
      }catch(IOException ex){
         System.out.println("Error writing file " + filename + ": " +
ex.getMessage());
      }//catch*/
   }//writePlotSkillCount

   private void writePlotSkillFailureCount(CSVFile SkillFailureCountResults){
      int max_x = getMaxTimeStep(SkillFailureCountResults);
      int min_x = 0;
      float max_y = SkillFailureCountResults.getMaxValue() +
SkillFailureCountResults.getMaxValue()*0.3f;
      if(max_y == 0.0f){
         max_y = 0.1f;
      }//if
      int min_y = 0;
      String filename = getAONReportDirectory() +
"/graph_skill_failure_count.plt";
      m_plotFileNames.add("graph_skill_failure_count.plt");
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(filename));
         out.write("set title \"AON Skill Failure Count\\n " +
m_reportDescription + "\"");out.newLine();
         out.write("set style data histogram");out.newLine();
         out.write("set style histogram cluster gap 5");out.newLine();
         out.write("set style fill solid border -1");out.newLine();
         out.write("set boxwidth 0.9");out.newLine();
         out.write("set xtic rotate by -45");out.newLine();
         out.write("set bmargin 10");out.newLine();
         out.write("set yrange [ 0.00000 : " + max_y + " ] noreverse
nowriteback");out.newLine();
         out.write("set datafile separator \",\"");out.newLine();
         if(m_numberOfSkills > 1){
            out.write("plot 'SkillFailureCountResults" + m_outputDirectory +
".csv' using 2:xtic(1) title col, \\");out.newLine();
         }else{
            out.write("plot 'SkillFailureCountResults" + m_outputDirectory +
".csv' using 2:xtic(1) title col");out.newLine();
         }//else
         for(int i = 3; i <= m_numberOfSkills+1; i++){
            if(i == 3){
               out.write("   '' u " + 3 + " ti col");
            }else{
               out.write(", '' u " + i + " ti col");
            }//else
         }//for
         out.newLine();
         out.write("set terminal emf  'Times Roman Bold' 18");out.newLine();
         out.write("set output \"" + m_outputDirectory +
"_SkillFailureCount.emf\"");out.newLine();
         out.write("replot");out.newLine();
         out.write("set output");out.newLine();
         out.write("set terminal windows");out.newLine();
```

```
         out.write("reset");out.newLine();
         out.close();
         System.out.println("Wrote file: " + filename);
      }catch(IOException ex){
         System.out.println("Error writing file " + filename + ": " +
ex.getMessage());
      }//catch*/
   }//writePlotSkillFailureCount

   private void writePlotSkillDemandCount(CSVFile SkillDemandCountResults){
      int max_x = getMaxTimeStep(SkillDemandCountResults);
      int min_x = 0;
      float max_y = SkillDemandCountResults.getMaxValue() +
SkillDemandCountResults.getMaxValue()*0.3f;
      if(max_y == 0.0f){
         max_y = 0.1f;
      }//if
      int min_y = 0;
      String filename = getAONReportDirectory() +
"/graph_skill_demand_count.plt";
      m_plotFileNames.add("graph_skill_demand_count.plt");
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(filename));
         out.write("set title \"AON Skill Demand Count\\n " +
m_reportDescription + "\"");out.newLine();
         out.write("set style data histogram");out.newLine();
         out.write("set style histogram cluster gap 5");out.newLine();
         out.write("set style fill solid border -1");out.newLine();
         out.write("set boxwidth 0.9");out.newLine();
         out.write("set xtic rotate by -45");out.newLine();
         out.write("set bmargin 10");out.newLine();
         out.write("set yrange [ 0.00000 : " + max_y + " ] noreverse
nowriteback");out.newLine();
         out.write("set datafile separator \",\"");out.newLine();
         if(m_numberOfSkills > 1){
            out.write("plot 'SkillDemandCountResults" + m_outputDirectory +
".csv' using 2:xtic(1) title col, \\");out.newLine();
         }else{
            out.write("plot 'SkillDemandCountResults" + m_outputDirectory +
".csv' using 2:xtic(1) title col");out.newLine();
         }//else
         for(int i = 3; i <= m_numberOfSkills+1; i++){
            if(i == 3){
               out.write("   '' u " + 3 + " ti col");
            }else{
               out.write(", '' u " + i + " ti col");
            }//else
         }//for
         out.newLine();
         out.write("set terminal emf  'Times Roman Bold' 18");out.newLine();
         out.write("set output \"" + m_outputDirectory +
"_SkillDemandCount.emf\"");out.newLine();
         out.write("replot");out.newLine();
         out.write("set output");out.newLine();
         out.write("set terminal windows");out.newLine();
         out.write("reset");out.newLine();
         out.close();
         System.out.println("Wrote file: " + filename);
      }catch(IOException ex){
         System.out.println("Error writing file " + filename + ": " +
ex.getMessage());
      }//catch*/
   }//writePlotSkillDemandCount
```

```
   private void writeInitialAgentCountFile(CSVFile AgentCountResults){
      int max_x = getMaxTimeStep(AgentCountResults);
      int min_x = 0;
      float max_y = AgentCountResults.getMaxValue() +
AgentCountResults.getMaxValue()*0.3f;
      if(max_y == 0.0f){
         max_y = 0.1f;
      }//if
      int min_y = 0;
      String filename = getAONReportDirectory() +
"/graph_initial_agent_count.plt";
      m_plotFileNames.add("graph_initial_agent_count.plt");
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(filename));
         out.write("set title \"AON Initial Agent Count\\n " +
m_reportDescription + "\"");out.newLine();
         out.write("set style data histogram");out.newLine();
         out.write("set style histogram cluster gap 5");out.newLine();
         out.write("set style fill solid border -1");out.newLine();
         out.write("set boxwidth 0.9");out.newLine();
         out.write("set xtic rotate by -45");out.newLine();
         out.write("set bmargin 10");out.newLine();
         out.write("set yrange [ 0.00000 : 110.00000 ] noreverse
nowriteback");out.newLine();
         out.write("set datafile separator \",\"");out.newLine();
         if(getLength(m_agentTypes) > 1){
            out.write("plot 'AgentCountResults" + m_outputDirectory + ".csv'
using 2:xtic(1) title col, \\");out.newLine();
         }else{
            out.write("plot 'AgentCountResults" + m_outputDirectory + ".csv'
using 2:xtic(1) title col");out.newLine();
         }//else
         for(int i = 3; i <= getLength(m_agentTypes); i++){
            if(i == 3){
               out.write("   '' u " + 3 + " ti col");
            }else{
               out.write(", '' u " + i + " ti col");
            }//else
         }//for
         out.newLine();
         out.write("set terminal emf  'Times Roman Bold' 18");out.newLine();
         out.write("set output \"" + m_outputDirectory +
"_InitialAgentCount.emf\"");out.newLine();
         out.write("replot");out.newLine();
         out.write("set output");out.newLine();
         out.write("set terminal windows");out.newLine();
         out.write("reset");out.newLine();
         out.close();
         System.out.println("Wrote file: " + filename);
      }catch(IOException ex){
         System.out.println("Error writing file " + filename + ": " +
ex.getMessage());
      }//catch*/
   }//writeInitialAgentCountFile

   private void writeAgentCountFile(CSVFile AgentCountResults){
      int max_x = getMaxTimeStep(AgentCountResults);
      int min_x = 0;
      float max_y = AgentCountResults.getMaxValue() +
AgentCountResults.getMaxValue()*0.3f;
      if(max_y == 0.0f){
         max_y = 0.1f;
```

```
      }//if
      int min_y = 0;
      String filename = getAONReportDirectory() + "/graph_agent_count.plt";
      m_plotFileNames.add("graph_agent_count.plt");
      try{
         BufferedWriter out = new BufferedWriter(new FileWriter(filename));
         out.write("set title \"AON Agent Count\\n " + m_reportDescription +
"\"");out.newLine();
         out.write("set style data histogram");out.newLine();
         out.write("set style histogram cluster gap 5");out.newLine();
         out.write("set style fill solid border -1");out.newLine();
         out.write("set boxwidth 0.9");out.newLine();
         out.write("set xtic rotate by -45");out.newLine();
         out.write("set bmargin 10");out.newLine();
         out.write("set yrange [ 0.00000 : 110.00000 ] noreverse
nowriteback");out.newLine();
         out.write("set datafile separator \",\"");out.newLine();
         if(getLength(m_agentTypes) > 1){
            out.write("plot 'AgentCountResults" + m_outputDirectory + ".csv'
using 2:xtic(1) title col, \\");out.newLine();
         }else{
            out.write("plot 'AgentCountResults" + m_outputDirectory + ".csv'
using 2:xtic(1) title col");out.newLine();
         }//else
         for(int i = 3; i <= getLength(m_agentTypes); i++){
            if(i == 3){
               out.write("   '' u " + 3 + " ti col");
            }else{
               out.write(", '' u " + i + " ti col");
            }//else
         }//for
         out.newLine();
         out.write("set terminal emf  'Times Roman Bold' 18");out.newLine();
         out.write("set output \"" + m_outputDirectory +
"_AgentCount.emf\"");out.newLine();
         out.write("replot");out.newLine();
         out.write("set output");out.newLine();
         out.write("set terminal windows");out.newLine();
         out.write("reset");out.newLine();
         out.close();
         System.out.println("Wrote file: " + filename);
      }catch(IOException ex){
         System.out.println("Error writing file " + filename + ": " +
ex.getMessage());
      }//catch*/
   }//writeAgentCountFile

   private void writePlotAgentTypeCountTimeSeriesFile(String testName, CSVFile
AgentCountResults){
      String plotName = "graph_" + testName + "_agent_count_time_series.plt";
      String filename = getAONReportDirectory() + "/" + plotName;
      m_plotFileNames.add(plotName);
      GnuplotFile gnuplot = new GnuplotFile();
      gnuplot.setTitle("AON " + testName + " Agent Count");
      gnuplot.setSubtitle(m_reportDescription);
      gnuplot.setXLabel("time");
      gnuplot.setYLabel("agent count");

gnuplot.plotYErrorBars(AgentCountResults,m_testNames,testName+"AgentCountResult
s","AgentCountTimeSeries",m_outputDirectory);
      gnuplot.save(filename);
   }//writePlotAgentTypeCountTimeSeriesFile
```

```
   private void writeAgentCountPieChart(CSVFile AgentTypeCountResults){
      PieChart pie = new PieChart();
      pie.setTitle("AON Agent Count");
      pie.setHeaderRow(0);
      pie.setValuesRow(1);
      pie.setStartColumn(1);

pie.createPieChart(getAONReportDirectory(),"AONAgentcount",AgentTypeCountResult
s);
   }//writeAgentCountPieChart
}//class ReportAON2

------------------------------ReportAON2.java-----------------------------------
-

------------------------------ReportAON.java------------------------------------
-

import java.util.*;
import java.io.File;

/**
 * ReportAON Class
 *
 *
 */
public class ReportAON implements Runnable{
   boolean m_insertFirstColumn;
                              /*
*/

   String m_inputFilename;   /*
*/
   String m_outputDirectory; /*
*/
   int m_numberOfInputFiles; /*
*/
   String m_description;     /*
*/
   int m_minAdaptTimeStep;   /*
*/
   Vector<String> m_agentTypes;

   /* constructor
*/
   public ReportAON(){
      m_insertFirstColumn = true;

      m_inputFilename = "";
      m_outputDirectory = "";
      m_numberOfInputFiles = 3;
      m_description = "";
      m_agentTypes = new Vector();
   }//constructor

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains
```

```
   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   public void setInputFilename(String inputFilename){
      m_inputFilename = inputFilename;
   }//setInputFilename

   public void setCountInput(int countInput){
      m_numberOfInputFiles = countInput;
   }//setCountInput

   public void setOutputDirectory(String outputDirectory){
      m_outputDirectory = outputDirectory;
   }//setOutputDirectory

   public void setDescription(String description){
      m_description = description;
   }//setDescription

   public static void main(String args[]){
      ReportAON report = new ReportAON();

      for(int i = 0; i < args.length; i++){
         report.processArg(i,args);
      }//for

      report.run();
   }//main

   private boolean processArg(int i, String args[]){
      try{
         if(args[i].compareTo("-input") == 0){
            m_inputFilename = args[i+1];
         }else if(args[i].compareTo("-output_directory") == 0){
            m_outputDirectory = args[i+1];
         }else if(args[i].compareTo("-count_input") == 0){
            m_numberOfInputFiles = Integer.parseInt(args[i+1]);
         }else if(args[i].compareTo("-description") == 0){
            m_description = args[i+1];
         }//else if
```

```
      }catch(Exception ex){
         System.out.println("Invalid argument");
         System.out.println("Usage: ReportAON [-output_directory <filename>] [-
count_input <number>] [-description <string>]");
         return false;
      }//catch
      return true;
   }//processArg

   public void run(){
      if(m_inputFilename != ""){
         readInputFile();
      }//if

      processReport();
   }//run

   public void processReport(){
      String directoryString = "";
      File directoryReport = new File(m_outputDirectory);
      if(directoryReport.exists()){
         directoryString = m_outputDirectory + "/";
      }//if

      System.out.println("output_directory=" + m_outputDirectory +
"\ncount_input=" + m_numberOfInputFiles + "\ndescription=" + m_description);

      AONReportModel file = new AONReportModel();

file.addCategory("optimal_performance",directoryString+"ReportAONOptimalPerform
ance_" + m_description + ".csv");
      file.addCategory("performance",directoryString+"ReportAONPerformance_" +
m_description + ".csv");
      file.addCategory("efficiency",directoryString+"ReportAONEfficiency_" +
m_description + ".csv");
      file.addCategory("changed_edges",directoryString+"ReportAONChangedEdges_"
+ m_description + ".csv");

file.addCategory("average_degree",directoryString+"ReportAONAverageDegree_" +
m_description + ".csv");

file.addCategory("average_density",directoryString+"ReportAONAverageDensity_" +
m_description + ".csv");
      file.addCategory("min_degree",directoryString+"ReportAONMinDegree_" +
m_description + ".csv");
      file.addCategory("max_degree",directoryString+"ReportAONMaxDegree_" +
m_description + ".csv");

file.addCategory("degree_variance",directoryString+"ReportAONDegreeVariance_" +
m_description + ".csv");
      file.addCategory("time_waiting",directoryString+"ReportAONTimeWaiting_" +
m_description + ".csv");
      file.addCategory("time_rewiring",directoryString+"ReportAONTimeRewiring_"
+ m_description + ".csv");

file.addCategory("time_proposing",directoryString+"ReportAONTimeProposing_" +
m_description + ".csv");
      file.addCategory("time_joining",directoryString+"ReportAONTimeJoining_" +
m_description + ".csv");
      file.addCategory("time_working",directoryString+"ReportAONTimeWorking_" +
m_description + ".csv");
      file.addCategory("total_utility",directoryString+"ReportAONTotalUtility_"
+ m_description + ".csv");
```

```
file.addCategory("changed_agents",directoryString+"ReportAONChangedAgents_" +
m_description + ".csv");

file.addCategory("average_task_distance",directoryString+"ReportAONTaskDistance
_" + m_description + ".csv");

file.addCategory("pending_failures",directoryString+"ReportAONPendingFailures_"
+ m_description + ".csv");

file.addCategory("skill_failures",directoryString+"ReportAONSkillFailures_" +
m_description + ".csv");

file.addCategory("distance_failures",directoryString+"ReportAONDistanceFailures
_" + m_description + ".csv");

file.addCategory("tasks_completed",directoryString+"ReportAONTasksCompleted_" +
m_description + ".csv");
      file.addCategory("task_clusters",directoryString+"ReportAONTaskClusters_"
+ m_description + ".csv");
      file.addCategory("task_count",directoryString+"ReportAONTaskCount_" +
m_description + ".csv");

file.addCategory("distance_travelled",directoryString+"ReportAONDistanceTravell
ed_" + m_description + ".csv");

file.addCategory("joined_impossible",directoryString+"ReportAONJoinedImpossible
_" + m_description + ".csv");

file.addCategory("proposed_impossible",directoryString+"ReportAONProposedImposs
ible_" + m_description + ".csv");
      file.addCategory("repeat_task",directoryString+"ReportAONRepeatTask_" +
m_description + ".csv");

file.addCategory("min_skill_shortage",directoryString+"ReportAONMinSkillShortag
e_" + m_description + ".csv");

file.addCategory("max_skill_shortage",directoryString+"ReportAONMaxSkillShortag
e_" + m_description + ".csv");

file.addCategory("average_skill_shortage",directoryString+"ReportAONAverageSkil
lShortage_" + m_description + ".csv");
      file.addCategory("alive_agents",directoryString+"ReportAONAliveAgents_" +
m_description + ".csv");
      for(int i = 0; i < getLength(m_agentTypes); i++){
         String agentType = (String)getItem(i,m_agentTypes);
         file.addCategory(agentType + "_count",directoryString+"ReportAON" +
agentType + "Count_" + m_description + ".csv");
//          file.addCategory(agentType + "_delta",directoryString+"ReportAON" +
agentType + "Delta_" + m_description + ".csv");
//          file.addCategory(agentType + "_utility",directoryString+"ReportAON"
+ agentType + "Utility_" + m_description + ".csv");
      }//for

      for(int i = 1; i <= m_numberOfInputFiles; i++){
         System.out.println("Read file: " + directoryString + "AONResults_" + i
+ m_description + ".csv");
         file.readFile(directoryString + "AONResults_" + i + m_description +
".csv");
      }//for

      System.out.println("Compute averages:");
      file.computeAveragePerRun();
```

```
      System.out.println("Compute confidence:");
      file.computeConfidence();
      System.out.println("Compute improvements:");
      file.computeImprovement(m_minAdaptTimeStep);
      file.save();

   }//processReport

   private void readInputFile(){
      System.out.println("readInputFile");
      AONXMLReader xmlFile = new AONXMLReader();
      if(!xmlFile.openXML(m_inputFilename)){
         System.out.println("Bad XML File: " + m_inputFilename + " Using
default AON settings");
         return;
      }//if

      XMLElement root = xmlFile.getRoot();
      String attributeName = "";
      String value = "";
      try{
         attributeName = "MinAdaptTimeStep";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_minAdaptTimeStep =
Integer.parseInt(value);

         attributeName = "OutputDirectory";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_outputDirectory = value;

         attributeName = "Description";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_description = value;

         System.out.println("Read AON parameters from " + m_inputFilename);
      }catch(Exception ex){
         System.out.println("Error loading attribute " + attributeName + "from
xml file: " + ex.getMessage());
      }//catch

      try{
         XMLElement xmlAgentTypes = root.getElement("AgentTypes");
         if(xmlAgentTypes != null){
            int agentTypesCount = xmlAgentTypes.getNumElements();
            for(int i = 0; i < agentTypesCount; i++){
               XMLElement xmlAgentType = xmlAgentTypes.getElement(i);
               String type = xmlAgentType.getAttribute("Name");
               String colorName = xmlAgentType.getAttribute("Color");
               m_agentTypes.add(type);
//                m_agentTypeAgentIndexLookup.add(new Vector());
//                m_agentTypeUtility.add(0.0f);
//                m_agentTypeCount.add(0);
//                m_agentTypeChangeCount.add(0);
//                m_agentTypeColor.put(type,colorName);
            }//for
         }//if
         System.out.println("Read Agent Types from " + m_inputFilename);
      }catch(Exception ex){
         System.out.println("Error reading Agent Types from xml file: " + ex);
      }//catch
   }//readInputFile
}//class ReportAON
```

```
------------------------------ReportAON.java-------------------------------------
-

---------------------------------Roads.java--------------------------------------
-

import java.util.*;
import java.awt.*;

import java.awt.image.*;
import javax.imageio.ImageIO;
import java.io.File;
import java.io.IOException;

/**
 * Roads Class
 *
 *
 */
public class Roads{
   static int ID_COL = 0, NAME_COL = 1, X1_COL = 2, Y1_COL = 3, X2_COL = 4,
Y2_COL = 5, PASSABLE_COL = 6;
   private CSVFile m_attributeTable;
                              /*
*/
   String m_filename;         /*
*/
   Hashtable m_intersections;/*
*/
   boolean m_firstDraw;       /*
*/
   BufferedImage m_image;     /*
*/
   String m_description;      /*
*/
   String m_outputDirectory; /*
*/
   int m_minX;                /*
*/
   int m_minY;                /*
*/
   int m_width;               /*
*/
   int m_height;              /*
*/
   boolean m_drawBackground; /*
*/

   /* constructor
*/
   public Roads(){
      m_attributeTable = new CSVFile();
      m_filename = "DefaultRoads.csv";
      m_intersections = new Hashtable();
      m_firstDraw = true;
      m_image = null;
      m_description = "DefaultRoad";
      m_outputDirectory = "";
      m_width = 100;
      m_height = 100;
      m_drawBackground = true;
   }//constructor
```

```
   /* constructor
*/
   public Roads(String filename){
      m_attributeTable = new CSVFile(filename);
      m_filename = filename;
      m_intersections = new Hashtable();
      m_firstDraw = true;
      m_image = null;
      m_description = "DefaultRoad";
      m_outputDirectory = "";
      m_width = 100;
      m_height = 100;
      m_drawBackground = true;
   }//constructor

   public void addLine(Point p1, Point p2, String name){
      if(m_attributeTable.getLineCount() == 0){
         String columnHeaders[] = {"id","name","x1","y1","x2","y2","passable"};
         m_attributeTable.addRow(columnHeaders);
      }//if
      String id = String.valueOf(m_attributeTable.getLineCount());
      String x1 = String.valueOf(p1.getX()), y1 = String.valueOf(p1.getY());
      String x2 = String.valueOf(p2.getX()), y2 = String.valueOf(p2.getY());
      String columnValues[] = {id,name,x1,y1,x2,y2,"true"};
      m_attributeTable.addRow(columnValues);
      addIntersection(x1,y1,id);
      addIntersection(x2,y2,id);
      m_firstDraw = true;
   }//addLine

   private void addIntersection(String x, String y, String id){
      String key = x + "-" + y;
      Vector lines = (Vector)m_intersections.get(key);
      if(lines != null){
         lines.add(id);
         m_intersections.put(key,lines);
      }else{
         Vector newLines = new Vector();
         newLines.add(id);
         m_intersections.put(key,newLines);
      }//else
   }//addIntersection

   public Vector getIntersectionShapes(int x, int y){
      String key = x + "-" + y;
      return (Vector)m_intersections.get(key);
   }//getIntersectionShapes

   public boolean open(){
      return open(m_filename);
   }//open

   public boolean open(String filename){
      m_filename = filename;
      if(m_attributeTable.open(filename)){
         m_description = filename.substring(0,filename.indexOf('.'));
         int lineCount = m_attributeTable.getLineCount()-1;
         for(int i = 1; i < lineCount; i++){
            int x1 = 0, y1 = 0, x2 = 0, y2 = 0;
            String id, x1String, y1String, x2String, y2String;
            id = m_attributeTable.getItem(i,ID_COL);
            x1String = m_attributeTable.getItem(i,X1_COL);
            y1String = m_attributeTable.getItem(i,Y1_COL);
```

```
            x2String = m_attributeTable.getItem(i,X2_COL);
            y2String = m_attributeTable.getItem(i,Y2_COL);
            //System.out.println("i=" + i + " id=" + id + " x1=" + x1String + "
y1=" + y1String + " x2=" + x2String + " y2=" + y2String);
            try{
               x1 = Integer.parseInt(x1String);
               y1 = Integer.parseInt(y1String);
               x2 = Integer.parseInt(x2String);
               y2 = Integer.parseInt(y2String);
               addIntersection(x1String,y1String,id);
               addIntersection(x2String,y2String,id);
               updateBounds(x1,y1);
               updateBounds(x2,y2);
            }catch(Exception ex){
               //ex.printStackTrace();
            }//catch
         }//for
         //System.out.println("Roads: " + filename + " Count Intersections: " +
m_intersections.size());
         m_firstDraw = true;
         return true;
      }else{
         return false;
      }//else
   }//open

   public boolean save(){
      return save(m_filename);
   }//save

   public boolean save(String filename){
      return m_attributeTable.save(filename);
   }//save

   public void close(){
      m_attributeTable.close();
   }//close

   private boolean loadImage(){
      String inputFilename = "";
      if(checkDirectory()){
         inputFilename = m_outputDirectory + "/" + m_description + ".jpg";
      }else{
         inputFilename = m_description + ".jpg";
      }//else
      try{
         m_image = ImageIO.read(new File(inputFilename));
      }catch(Exception ex){
         //ex.printStackTrace();
         System.out.println("Image for roads file " + m_description + ".csv not
found:  Image will be created");
         return false;
      }//catch
      return true;
   }//loadImage

   public void paint(Graphics g, int x_offset, int y_offset){
      if(m_firstDraw){
         if(!loadImage()){
            paintImage(g,0,0);
         }//if
         m_firstDraw = false;
      }//if
```

```
      g.drawImage(m_image,x_offset,y_offset,null);
   }//paint

   private void paintRoads(Graphics g, int x_offset, int y_offset){
      if(m_drawBackground){
         g.setColor(Color.white);
         g.fillRect(0,0,m_width,m_height);
      }//if
      int lineCount = m_attributeTable.getLineCount();
      for(int i = 1; i < lineCount; i++){
         int x1, y1, x2, y2;
         x1 = Integer.parseInt(m_attributeTable.getItem(i,X1_COL));
         y1 = Integer.parseInt(m_attributeTable.getItem(i,Y1_COL));
         x2 = Integer.parseInt(m_attributeTable.getItem(i,X2_COL));
         y2 = Integer.parseInt(m_attributeTable.getItem(i,Y2_COL));
         boolean passable =
parseBoolean(m_attributeTable.getItem(i,PASSABLE_COL));

         if(passable){
            g.setColor(Color.magenta);
         }else{
            g.setColor(Color.pink);
         }//else
         g.drawLine(x_offset+x1,y_offset+y1,x_offset+x2,y_offset+y2);
      }//for
   }//paint

   private void paintImage(Graphics g, int x_offset, int y_offset){
      String springDescription = "";
      Rectangle captureSize = new Rectangle(this.getBounds());
      //BufferedImage bufferedImage = null;

      m_image = new
BufferedImage((int)captureSize.getWidth(),(int)captureSize.getHeight(),Buffered
Image.TYPE_INT_RGB);
      paintRoads((Graphics)m_image.getGraphics(),x_offset,y_offset);

      try{
      //      String names[] = ImageIO.getWriterFormatNames();
      //      for(int i = 0; i < names.length; i++){
      //        System.out.print(" " + names[i]);
      //      }//for
         String outputFilename = "";
         if(checkDirectory()){
            outputFilename = m_outputDirectory + "/" + m_description + ".jpg";
         }else{
            outputFilename = m_description + ".jpg";
         }//else
         ImageIO.write(((RenderedImage)m_image), "JPEG", new
File(outputFilename));
      }catch(IOException ex){
         System.out.println("Error writing roads image: " + ex);
      }//catch
   }//paintImage

   private boolean parseBoolean(String value){
      if(value.compareToIgnoreCase("true") == 0){
        return true;
      }else{
        return false;
      }//else
   }//parseBoolean
```

```
   public int getShape(Point p1, Point p2){
      Vector<String> shapes1 =
getIntersectionShapes((int)p1.getX(),(int)p1.getY());
      Vector<String> shapes2 =
getIntersectionShapes((int)p2.getX(),(int)p2.getY());
      for(int i = 0; i < shapes1.size(); i++){
         String shape1_id = shapes1.elementAt(i);
         if(shapes2.contains(shape1_id)){
            return Integer.parseInt(shape1_id);
         }//if
      }//for
      return -1;
   }//getShape

   public int getRandomShape(){
      Random rand = new Random();
      int shapeID = rand.nextInt(m_attributeTable.getLineCount()-1) + 1;
      //System.out.println("shapeID=" + shapeID);
      return shapeID;
   }//getRandomShape

   public int getNearestShape(Point p){
      int min_dist = 100000;
      int min_shape_id = -1, x = (int)p.getX(), y = (int)p.getY();
      for(int i = 1; i <= m_attributeTable.getLineCount() - 1; i++){
         int x1, y1, x2, y2;
         x1 = Integer.parseInt(m_attributeTable.getItem(i,X1_COL));
         y1 = Integer.parseInt(m_attributeTable.getItem(i,Y1_COL));
         x2 = Integer.parseInt(m_attributeTable.getItem(i,X2_COL));
         y2 = Integer.parseInt(m_attributeTable.getItem(i,Y2_COL));

         int distance = getDistance(x1,y1,x,y);
         if(distance < min_dist){
            min_shape_id = i;
            min_dist = distance;
         }//if
         distance = getDistance(x2,y2,x,y);
         if(distance < min_dist){
            min_shape_id = i;
            min_dist = distance;
         }//if
      }//for

      return min_shape_id;
   }//getNearestShape

   public Point getRandomLocation(int id){
      Random rand = new Random();
      int x1, y1, x2, y2;
      x1 = Integer.parseInt(m_attributeTable.getItem(id,X1_COL));
      y1 = Integer.parseInt(m_attributeTable.getItem(id,Y1_COL));
      x2 = Integer.parseInt(m_attributeTable.getItem(id,X2_COL));
      y2 = Integer.parseInt(m_attributeTable.getItem(id,Y2_COL));

      int length_x = x1 - x2, length_y = y1 - y2;

      float scale = rand.nextFloat();
      return new Point(x2+(int)(scale*length_x),y2+(int)(scale*length_y));
   }//getRandomLocation

   public Point getRandomTaskLocation(int taskRange){
      Random rand = new Random();
      int id = getRandomShape();
```

```
      int x1, y1;
      x1 = Integer.parseInt(m_attributeTable.getItem(id,X1_COL));
      y1 = Integer.parseInt(m_attributeTable.getItem(id,Y1_COL));
      int sign = 1;
      if(rand.nextFloat() < 0.5f)sign = -1;
      return new
Point(x1+sign*rand.nextInt(taskRange/2),y1+sign*rand.nextInt(taskRange/2));
   }//getRandomTaskLocation

   public Point getShapeIntersection1(int id){
      int x1, y1;
      x1 = Integer.parseInt(m_attributeTable.getItem(id,X1_COL));
      y1 = Integer.parseInt(m_attributeTable.getItem(id,Y1_COL));
      return new Point(x1,y1);
   }//getShapeIntersections

   public Point getShapeIntersection2(int id){
      int x2, y2;
      x2 = Integer.parseInt(m_attributeTable.getItem(id,X2_COL));
      y2 = Integer.parseInt(m_attributeTable.getItem(id,Y2_COL));
      return new Point(x2,y2);
   }//getShapeIntersection2

   public Vector<Point> getPlan(int startX, int startY, int shapeID, int goalX,
int goalY, int precision, int maxDepth){
      int currentDepth = 0;
      Vector<String> visited = new Vector();
      PriorityQueue queue = new PriorityQueue();

      Point startP1 = getShapeIntersection1(shapeID);
      Vector<Point> startActionsP1 = new Vector();
      startActionsP1.add(startP1);
      int startDistance1 = getDistance(startP1,new Point(startX,startY));
      PlanItem item1 = new PlanItem(startP1,startActionsP1,startDistance1);
      queue.add(item1,startDistance1 + getDistance(new
Point(goalX,goalY),startP1));
      visited.add(startP1.toString());

      Point startP2 = getShapeIntersection2(shapeID);
      Vector<Point> startActionsP2 = new Vector();
      startActionsP2.add(startP2);
      int startDistance2 = getDistance(startP2,new Point(startX,startY));
      PlanItem item2 = new PlanItem(startP2,startActionsP2,startDistance2);
      queue.add(item2,startDistance2 + getDistance(new
Point(goalX,goalY),startP2));
      visited.add(startP2.toString());

      while(queue.size() > 0){
         PlanItem planItem = (PlanItem)queue.remove();
         int goalDistance =
getDistance((int)planItem.point.getX(),(int)planItem.point.getY(),goalX,goalY);
         if(goalDistance <= precision){// || planItem.actions.size() >= 10){ ||
currentDepth >= maxDepth){
            //System.out.println("PLAN: " + planItem.actions);
            //System.out.println("Plan length=" + planItem.actions.size() + "
depth reached=" + currentDepth);
            return planItem.actions;
         }//if

         Vector<String> shapes =
getIntersectionShapes((int)planItem.point.getX(),(int)planItem.point.getY());
         if(shapes != null){
            for(int i = 0; i < shapes.size(); i++){
```

```
               int id = Integer.parseInt(shapes.elementAt(i));
               Point p1 = getShapeIntersection1(id);
               Point p2 = getShapeIntersection2(id);
               if(!visited.contains(p1.toString())){
                  Vector<Point> a_p1 = (Vector)planItem.actions.clone();
                  a_p1.add(p1);
                  int distance = getDistance(planItem.point,p1) +
planItem.distance;
                  PlanItem nextItem = new PlanItem(p1,a_p1,distance);
                  queue.add(nextItem,distance + getDistance(new
Point(goalX,goalY),p1));
                  visited.add(p1.toString());
               }//if
               if(!visited.contains(p2.toString())){
                  Vector<Point> a_p2 = (Vector)planItem.actions.clone();
                  a_p2.add(p2);
                  int distance = getDistance(planItem.point,p2) +
planItem.distance;
                  PlanItem nextItem = new PlanItem(p2,a_p2,distance);
                  queue.add(nextItem,distance + getDistance(new
Point(goalX,goalY),p2));
                  visited.add(p2.toString());
               }//if
            }//for
         }//if

         currentDepth++;
//          if(currentDepth > 300){
//             System.err.println("Possible infinite loop: Roads.getPlan() ->
currentDepth=" + currentDepth);
//             System.out.println("Possible infinite loop: Roads.getPlan() ->
currentDepth=" + currentDepth);
//          }//if
      }//while
      System.out.println("NO PLAN FOUND: depth reached=" + currentDepth);
      return new Vector();
   }//getPlan

   private int getDistance(Point p1, Point p2){
      return
getDistance((int)p1.getX(),(int)p1.getY(),(int)p2.getX(),(int)p2.getY());
   }//getDistance

   private int getDistance(int x1, int y1, int x2, int y2){
      return (int)Math.sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
   }//getDistance

   private boolean checkDirectory(){
      File outDirectory = new File(m_outputDirectory);
      if(!outDirectory.isDirectory()){
        return outDirectory.mkdir();
      }//if
      return outDirectory.exists();
   }//checkDirectory

   public void setDrawBackground(boolean drawBackground){
      m_drawBackground = drawBackground;
   }//setDrawBackground

   public Rectangle getBounds(){
      return new Rectangle(m_width,m_height);
   }//getBounds
```

```
   private void updateBounds(int x, int y){
      if(x > m_width){
         m_width = x + 50;
      }//if
      if(y > m_height){
         m_height = y + 50;
      }//if
   }//updateBounds

   private boolean checkBounds(int x, int y){
      if((x >= 0 && x < m_width) && (y >= 0 && y < m_height)){
         return true;
      }else{
         return false;
      }//else
   }//checkBounds

   public void crop(int startX, int startY, int width, int height){
      m_intersections.clear();

      m_minX = startX;
      m_minY = startY;
      m_width = width;
      m_height = height;

      CSVFile newAttributeTable = new CSVFile();
      String columnHeaders[] = {"id","name","x1","y1","x2","y2","passable"};
      newAttributeTable.addRow(columnHeaders);

      int nextID = 1;
      int lineCount = m_attributeTable.getLineCount();
      for(int i = 1; i < lineCount; i++){
         String id, name, x1String, y1String, x2String, y2String, passable;
         id = m_attributeTable.getItem(i,ID_COL);
         name = m_attributeTable.getItem(i,NAME_COL);
         x1String = m_attributeTable.getItem(i,X1_COL);
         y1String = m_attributeTable.getItem(i,Y1_COL);
         x2String = m_attributeTable.getItem(i,X2_COL);
         y2String = m_attributeTable.getItem(i,Y2_COL);
         passable = m_attributeTable.getItem(i,PASSABLE_COL);
         int x1 = 0, y1 = 0, x2 = 0, y2 = 0;
         try{
            x1 = Integer.parseInt(x1String);
            y1 = Integer.parseInt(y1String);
            x2 = Integer.parseInt(x2String);
            y2 = Integer.parseInt(y2String);

            if(checkBounds(x1,y1) && checkBounds(x2,y2)){
               String rowValues[] = {String.valueOf(nextID), name, x1String,
y1String, x2String, y2String, passable};
               newAttributeTable.addRow(rowValues);
               addIntersection(x1String,y1String,String.valueOf(nextID));
               addIntersection(x2String,y2String,String.valueOf(nextID));
               nextID++;
            }//if
         }catch(Exception ex){
         }//catch
      }//for

      m_attributeTable = newAttributeTable;
   }//crop

   private class PlanItem{
```

```
      public Point point;    /*
*/
      public Vector<Point> actions;
      public int distance;   /*
*/

      /* constructor
*/
      public PlanItem(){
         point = new Point();
         actions = new Vector();
         distance = 0;
      }//constructor

      /* constructor
*/
      public PlanItem(Point p, Vector<Point> a, int d){
         point = p;
         actions = a;
         distance = d;
      }//constructor
   }//PlanItem
}//Roads
```

```
--------------------------------Roads.java-----------------------------------
-
```

```
-----------------------------RunTestAON.java---------------------------------
-

import java.lang.InterruptedException;
import java.util.Vector;
import java.util.Date;

/**
 * RunTestAON Class
 *
 *
 */
public class RunTestAON implements Runnable{
   int m_numberOfRuns;       /*
*/
   String m_inputFilename;   /*
*/

   int m_numberOfTests;      /*
*/
   Vector <String>m_testNames;
   Vector <String>m_adaptationNames;
   Vector <String>m_outputDirectories;
   Vector <String>m_descriptions;
   Vector <Integer>m_maxNodeDegrees;
   Vector <Integer>m_communicationDepths;
   Vector <Integer>m_neighborhoodRadius;
   Vector <Boolean>m_useCommandSkills;
   Vector <Integer>m_taskRange;
   boolean m_useTaskRange;   /*
*/
   Vector <String>m_agentTypes;
```

```
   /* constructor
*/
   public RunTestAON(){
      m_numberOfRuns = 0;
      m_inputFilename = "";

      m_testNames = new Vector();
      m_adaptationNames = new Vector();
      m_outputDirectories = new Vector();
      m_descriptions = new Vector();
      m_maxNodeDegrees = new Vector();
      m_communicationDepths = new Vector();
      m_neighborhoodRadius = new Vector();
      m_useCommandSkills = new Vector();
      m_taskRange = new Vector();
      m_useTaskRange = false;
      m_agentTypes = new Vector();
   }//constructor

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   public void setInputFilename(String inputFilename){
      m_inputFilename = inputFilename;
   }//setInputFilename

   public static void main(String args[]){
      RunTestAON runTest = new RunTestAON();
      for(int i = 0; i < args.length; i++){
         runTest.processArg(i,args);
      }//for
```

```
      runTest.run();
   }//main

   private boolean processArg(int i, String args[]){
      try{
         if(args[i].compareTo("-input") == 0){
            m_inputFilename = args[i+1];
         }else if(args[i].compareTo("-run") == 0){
            m_numberOfRuns = Integer.parseInt(args[i+1]);
         }//else if
      }catch(Exception ex){
         System.out.println("Invalid argument");
         System.out.println("Usage: TestAON -input <string>[the filename of the
xml settings input file]");
         System.out.println("               -run <number>[the number of
iterations to run the test]");
         return false;
      }//catch
      return true;
   }//processArg

   public void run(){
      Date current1 = new Date(System.currentTimeMillis());
      System.out.println("----------------------------  RunTestAON  -----------
-----------------");
      System.out.println("Start: " + current1);
      if(m_inputFilename != ""){
         readInputFile();
         try{
            validateAgentTypes();
            validateTestOutputDirectories();
         }catch(Exception ex){
            System.out.print(ex.toString());
            System.err.println(ex.toString());
            return;
         }//catch
      }//if

      System.out.println("----------- Test Parameters ---------------- ");
      System.out.println("Test names: " + m_testNames);
      System.out.println("Agent types: " + m_adaptationNames);
      System.out.println("Number of tests=" + m_numberOfTests);
      System.out.println("Number of runs=" + m_numberOfRuns);
      System.out.println("CommunicationDepths=" + m_communicationDepths);
      System.out.println("NeighborhoodRadius=" + m_neighborhoodRadius);
      System.out.println("UseCommandSkills=" + m_useCommandSkills);
      System.out.println("----------- End Test Parameters ----------- ");

      for(int i = 0; i < m_numberOfTests; i++){
         String testName = (String)getItem(i,m_testNames);
         String adaptationName = (String)getItem(i,m_adaptationNames);
         String outputDirectory = (String)getItem(i,m_outputDirectories);
         String description = (String)getItem(i,m_descriptions);
         int maxNodeDegree = ((Integer)getItem(i,m_maxNodeDegrees)).intValue();
         int communicationDepth =
((Integer)getItem(i,m_communicationDepths)).intValue();
         int neighborhoodRadius =
((Integer)getItem(i,m_neighborhoodRadius)).intValue();
         boolean useCommandSkills =
((Boolean)getItem(i,m_useCommandSkills)).booleanValue();
         int taskRange = ((Integer)getItem(i,m_taskRange)).intValue();
```

```
         System.out.println("----------------" + testName + "----------------
");
         System.out.println("outputDirectory=" + outputDirectory + "
description=" + description);
         for(int j = 1; j <= m_numberOfRuns; j++){
            System.out.println("----------------START RUN " + j + " " +
testName + "----------------");
            TestAON test = new TestAON();
            test.setAdaptation(adaptationName);
            test.setOutputDirectory(outputDirectory);
            test.setDescription(description);
            test.setMaxNodeDegree(maxNodeDegree);
            test.setCommunicationDepth(communicationDepth);
            test.setNeighborhoodRadius(neighborhoodRadius);
            test.setUseCommandSkills(useCommandSkills);
            if(m_useTaskRange){
               test.setTaskRange(taskRange);
            }//if
            test.setInputFilename(m_inputFilename);
            test.setRunNumber(j);
            startThread(test);
            System.out.println("-----------------END RUN " + j + " " + testName
+ "-----------------");
         }//for

         //ReportAON
         ReportAON report = new ReportAON();
         report.setInputFilename(m_inputFilename);
         report.setCountInput(m_numberOfRuns);
         report.setOutputDirectory(outputDirectory);
         report.setDescription(description);
         startThread(report);
      }//for

      if(m_numberOfTests > 0){
         //ReportAON2
         ReportAON2 report2 = new ReportAON2();
         report2.setInputFilename(m_inputFilename);
         report2.setCountInput(m_numberOfRuns);
         startThread(report2);
      }//if
      Date current2 = new Date(System.currentTimeMillis());
      System.out.println("Stop: " + current2);
      System.out.println("--------------------------- End RunTestAON  ----------
-----------------");
   }//run

   private void startThread(Runnable threadTarget){
      Thread t = new Thread(threadTarget);
      t.run();
      try{
         t.join();
      }catch(InterruptedException ex){
         System.out.println("Thread interrupted: " + ex);
      }//catch
   }//startThread

   private void readInputFile(){
      System.out.println("readInputFile");
      AONXMLReader xmlFile = new AONXMLReader();
      if(!xmlFile.openXML(m_inputFilename)){
         System.out.println("Bad XML File: " + m_inputFilename + " Using
default AON settings");
```

```
         return;
      }//if

      XMLElement root = xmlFile.getRoot();
      String attributeName = "";
      String value = "";

      XMLElement xmlTests = root.getElement("Tests");
      m_numberOfTests = xmlTests.getNumElements();
      for(int i = 0; i < m_numberOfTests; i++){
         String testName = "";
         String adaptationName = "";
         String outputDirectory = "";
         String description = "";
         int maxNodeDegree = 1000;
         int communicationDepth = 2;
         int neighborhoodRadius = 1;
         boolean useCommandSkills = false;
         int taskRange = 20;
         XMLElement xmlTest = xmlTests.getElement(i);
         try{
            attributeName = "Name";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)testName = value;

            attributeName = "Adaptation";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)adaptationName = value;

            attributeName = "OutputDirectory";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)outputDirectory = value;

            attributeName = "Description";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)description = value;

            attributeName = "MaxNodeDegree";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)maxNodeDegree =
Integer.parseInt(value);

            attributeName = "CommunicationDepth";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)communicationDepth =
Integer.parseInt(value);

            attributeName = "NeighborhoodRadius";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)neighborhoodRadius =
Integer.parseInt(value);

            attributeName = "UseCommandSkills";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)useCommandSkills =
parseBoolean(value);

            attributeName = "TaskRange";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null){
               m_useTaskRange = true;
               taskRange = Integer.parseInt(value);
            }//if
```

```
            System.out.println("Read AON parameters from " + m_inputFilename);
         }catch(Exception ex){
            System.out.println("Error loading test attribute " + attributeName
+ "from xml file: " + ex.getMessage());
         }//catch
         System.out.println("Name=" + testName + " directory=" +
outputDirectory + " description=" + description);
         m_testNames.add(testName);
         m_adaptationNames.add(adaptationName);
         m_outputDirectories.add(outputDirectory);
         m_descriptions.add(description);
         m_maxNodeDegrees.add(maxNodeDegree);
         m_communicationDepths.add(communicationDepth);
         m_neighborhoodRadius.add(neighborhoodRadius);
         m_useCommandSkills.add(useCommandSkills);
         m_taskRange.add(taskRange);
      }//for

      try{
         XMLElement xmlAgentTypes = root.getElement("AgentTypes");
         if(xmlAgentTypes != null){
            int agentTypesCount = xmlAgentTypes.getNumElements();
            for(int i = 0; i < agentTypesCount; i++){
               XMLElement xmlAgentType = xmlAgentTypes.getElement(i);
               String type = xmlAgentType.getAttribute("Name");
               m_agentTypes.add(type);
            }//for
            System.out.println("Read Agent Types from " + m_inputFilename);
         }//if
      }catch(Exception ex){
         System.out.println("Error reading Agent Types from xml file: " + ex);
         ex.printStackTrace();
      }//catch
   }//readInputFile

   private void validateAgentTypes() throws java.lang.Exception{
      System.out.println("Validate Agent Types and Adaptation names for tests
in XML file: \n  AgentTypes: " + m_agentTypes + "\n  Adaptations: " +
m_adaptationNames);
      for(int i = 0; i < getLength(m_adaptationNames); i++){
         String adaptation = (String)getItem(i,m_adaptationNames);
         if(adaptation != "" && adaptation != null){
            if(!contains(adaptation,m_agentTypes)){
               throw new Exception("\n    RunTestAON: Test adaptation " +
adaptation + "\n    Not included in Agent Types \n    XML Settings file " +
m_inputFilename);
            }//if
         }//if
      }//for
   }//validateAgentTypes

   private void validateTestOutputDirectories() throws java.lang.Exception{
      System.out.println("Validate unique Output Directories for tests in XML
file: \n: " + m_outputDirectories);
      Vector directories = new Vector();
      for(int i = 0; i < getLength(m_outputDirectories); i++){
         String outputDirectory = (String)getItem(i,m_outputDirectories);
         if(directories.contains(outputDirectory)){
            throw new Exception("\n     Duplicate output directory name\n
for tests in XML file: \n     " + outputDirectory);
         }//if
         directories.add(outputDirectory);
```

```
      }//for
   }//validateTestOutputDirectories

   private boolean parseBoolean(String value){
      if(value.compareToIgnoreCase("true") == 0){
        return true;
      }else{
        return false;
      }//else
   }//parseBoolean
}//RunTestAON

-----------------------------RunTestAON.java-----------------------------------
-

-------------------------SkillPriorityItem.java--------------------------------
-

import java.util.*;

/**
 * SkillPriorityItem Class
 *
 *
 */
public class SkillPriorityItem{
   int m_skill;              /*
*/
   Vector<Integer> m_agents;
   Vector<Integer> m_agentTaskCommittedTimeLeft;

   /* constructor
*/
   public SkillPriorityItem(int skill){
      m_skill = skill;
      m_agents = new Vector();
      m_agentTaskCommittedTimeLeft = new Vector();
   }//constructor

   public int getSkillID(){
      return m_skill;
   }//getSkillID

   public void addAgent(int agentID){
      m_agents.add(new Integer(agentID));
      m_agentTaskCommittedTimeLeft.add(new Integer(0));
   }//addAgent

   public boolean containsAgent(int agentID){
      return m_agents.contains(new Integer(agentID));
   }//containsSkill

   public Vector<Integer> getAgents(){
      return m_agents;
   }//getAgents

   public boolean agentAvailable(int agentID){
      int index = m_agents.indexOf(agentID);
      if(index >= 0){
         return (m_agentTaskCommittedTimeLeft.elementAt(index) <= 0);
      }//if
      return false;
   }//agentAvailable
```

```
   public int getAvailableAgent(){
      for(int i = 0; i < m_agents.size(); i++){
         if(m_agentTaskCommittedTimeLeft.elementAt(i) <= 0){
            return m_agents.elementAt(i);
         }//if
      }//for
      return -1;
   }//getAvailableAgent

   public void setAgentTimeLeft(int agentID, int timeLeft){
      int index = m_agents.indexOf(agentID);
      m_agentTaskCommittedTimeLeft.setElementAt(timeLeft,index);
   }//setAgentFree

   public void freeAllAgents(){
      for(int i = 0; i < m_agents.size(); i++){
         m_agentTaskCommittedTimeLeft.setElementAt(0,i);
      }//for
   }//freeAllAgents

   public void updateAllAgents(){
      for(int i = 0; i < m_agents.size(); i++){
         int timeLeft = m_agentTaskCommittedTimeLeft.elementAt(i);
         if(timeLeft > 0){
            m_agentTaskCommittedTimeLeft.setElementAt(timeLeft-1,i);
         }//if
      }//for
   }//updateAllAgents

   public int agentCount(){
      return m_agents.size();
   }//agentCount

   public int availableAgentCount(){
      int count = 0;
      for(int i = 0; i < m_agents.size(); i++){
         if(m_agentTaskCommittedTimeLeft.elementAt(i) <= 0){
            count++;
         }//if
      }//for
      return count;
   }//availableAgentCount

   public String toString(){
      String result = "SkillPriorityItem=[skill=" + m_skill + ",agents={";
      for(int i = 0; i < m_agents.size(); i++){
         if(i == 0){
            result = result + "id:" + m_agents.elementAt(i) + " timeLeft:" +
m_agentTaskCommittedTimeLeft.elementAt(i);
         }else{
            result = result + ", id:" + m_agents.elementAt(i) + " timeLeft:" +
m_agentTaskCommittedTimeLeft.elementAt(i);
         }//else
      }//for
      result = result + "}]";
      return result;
   }//toString
}//SkillPriorityItem

------------------------SkillPriorityItem.java-------------------------------
-
```

```
-------------------------SkillPriorityList.java--------------------------------
-

import java.util.Vector;

/**
 * SkillPriorityList Class
 *  This class stores the skills and the agents who possess the skill.
 *  The skillPriorityItems in the list are ordered by number of
 *  agents possessing the skill, in ascending order.
 *
 */
public class SkillPriorityList{
   Vector<SkillPriorityItem> m_skillList;

   /* constructor
*/
   public SkillPriorityList(){
      m_skillList = new Vector();
   }//constructor

   public boolean contains(int skillID){
      for(int i = 0; i < m_skillList.size(); i++){
         if(skillID == m_skillList.elementAt(i).getSkillID()){
            return true;
         }//if
      }//for
      return false;
   }//contains

   public SkillPriorityItem get(int skillID){
      int index = getIndex(skillID);
      if(index >= 0){
         return m_skillList.elementAt(index);
      }else{
         return null;
      }//else
   }//get

   public SkillPriorityItem remove(int skillID){
      SkillPriorityItem returnItem;
      int index = getIndex(skillID);
      if(index >= 0){
         returnItem = m_skillList.elementAt(index);
         m_skillList.removeElementAt(index);
      }else{
         returnItem = null;
      }//else
      return returnItem;
   }//remove

   public void put(int skillID, SkillPriorityItem item){
      int index = getIndex(skillID);
      if(index >= 0){
         m_skillList.removeElementAt(index);
      }//if

      boolean itemInserted = false;
      for(int i = 0; i < m_skillList.size() && !itemInserted; i++){
         SkillPriorityItem current = m_skillList.elementAt(i);
         if(item.agentCount() <= current.agentCount()){
            m_skillList.insertElementAt(item,i);
            itemInserted = true;
```

```
         }//if
      }//for

      if(itemInserted == false){
         m_skillList.add(item);
      }//if
   }//put

   private int getIndex(int skillID){
      for(int i = 0; i < m_skillList.size(); i++){
         if(skillID == m_skillList.elementAt(i).getSkillID()){
            return i;
         }//if
      }//for
      return -1;
   }//getIndex

   public void freeAllListAgents(){
      for(int i = 0; i < m_skillList.size(); i++){
         SkillPriorityItem item = m_skillList.elementAt(i);
         item.freeAllAgents();
      }//for
   }//freeAllAgents

   public void updateAllListAgents(){
      for(int i = 0; i < m_skillList.size(); i++){
         SkillPriorityItem item = m_skillList.elementAt(i);
         item.updateAllAgents();
      }//for
   }//updateAllAgents

   public SkillPriorityItem getItem(int index){
      if(index >= 0){
         return m_skillList.elementAt(index);
      }else{
         return null;
      }//else
   }//getItem

   public int size(){
      return m_skillList.size();
   }//size

   public String toString(){
      String result = "SkillPriorityList=[";
      for(int i = 0; i < m_skillList.size(); i++){
         if(i == 0){
            result = result + m_skillList.elementAt(i);
         }else{
            result = result + "," + m_skillList.elementAt(i);
         }//else
      }//for
      result = result + "]";
      return result;
   }//toString
}//SkillPriorityList

-------------------------SkillPriorityList.java--------------------------------
-

-----------------------SkillShortageHistory.java-----------------------------
-
```

```
import java.util.*;

/**
 * SkillShortageHistory Class
 *
 *
 */
public class SkillShortageHistory{
   int m_skillCount;         /*
*/
   int m_historyLength;      /*
*/
   Vector<Vector<Integer>> m_skillShortages;
   int m_historyIndex;       /*
*/

   /* constructor
*/
   public SkillShortageHistory(int skillCount, int historyLength){
      m_skillCount = skillCount;
      m_historyLength = historyLength;
      m_skillShortages = new Vector();
      for(int i = 0; i < m_historyLength; i++){
         Vector<Integer> skillShortage = new Vector();
         for(int j = 0; j < skillCount; j++){
            skillShortage.add(0);
         }//for
         m_skillShortages.add(skillShortage);
      }//for
      m_historyIndex = 0;
   }//constructor

   public int getSkillShortage(int skillID){
      Vector<Integer> skillShortage = new Vector();
      int shortageCount = 0;
      for(int i = 0; i < m_historyLength; i++){
         skillShortage = m_skillShortages.elementAt(i);
         shortageCount = shortageCount + skillShortage.elementAt(skillID);
      }//for
      return shortageCount;
   }//getSkillShortage

   public void addSkillShortage(Vector<Integer> skillShortage){
      m_skillShortages.set(m_historyIndex,skillShortage);
      m_historyIndex++;
      if(m_historyIndex >= m_historyLength){
         m_historyIndex = 0;
      }//if
   }//addSkillShortage
}//class SkillShortageHistory

-----------------------SkillShortageHistory.java-------------------------------
-

--------------------------StructuralAgent.java---------------------------------
-

import java.util.*;

/**
 * StructuralAgent Class
 *   This is an agent that seeks to rewire to agents who have the most
 *   networn connections
```

```
 */
public class StructuralAgent extends AgentX{
   /* constructor
*/
   public StructuralAgent(TestAON p,int id, int x, int y, Vector agents){
      super(p,id,x,y,"Structural",agents);
   }//constructor

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "StructuralAgent\n" + super.toString() + "\n";
      return value;
   }//toString
}//class StructuralAgent

--------------------------StructuralAgent.java---------------------------------
-

----------------------StructuralImpatientAgent.java-----------------------------
-

import java.util.*;

/**
 * StructuralImpatientAgent Class
 *   This is an agent that seeks to rewire to agents who have the most
 *   networn connections
 *   -This agent is 'impatient' because it builds off the Structural agent
 *     by choosing to drop its commitment to a task when the skills left to be
 *     filled in the task exceed the number of neighboring agents who are not
 *     committed to a task
 */
public class StructuralImpatientAgent extends StructuralAgent {
   /* constructor
*/
   public StructuralImpatientAgent(TestAON p,int id, int x, int y, Vector
agents){
      super(p,id,x,y,agents);
      m_type = "StructuralImpatient";
   }//constructor

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      boolean dropTask = false;
      if(getUnCommittedNeighborCount() < task.unFilledSkillsCount()){
         dropTask = true;
      }//if
      return dropTask;
   }//chooseToDropTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "StructuralImpatientAgent\n" + super.toString() + "\n";
      return value;
   }//toString
}//class StructuralImpatientAgent
```

```
----------------------StructuralImpatientAgent.java-----------------------------
-

-----------------------StructuralStratAgent.java-------------------------------
-

import java.util.*;

/**
 * StructuralStratAgent Class
 *   This is an agent that seeks to rewire to agents who have the most
 *   networn connections
 *   -This agent is 'strategic' because it builds off the Structural agent
 *    by choosing tasks to join and tasks to propose that are the best match
 *    for the skills possessed by the agent's uncommitted neighbors
 */
public class StructuralStratAgent extends StructuralAgent {
   /* constructor
*/
   public StructuralStratAgent(TestAON p,int id, int x, int y, Vector agents){
      super(p,id,x,y,agents);
      m_type = "StructuralStrat";
   }//constructor

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
       m_edgesChanged = 0;
       writeLog(m_id + " updateUncommitted:");
       // Look at all tasks which your immediate neighbor has committed to
       // Select the one which is best based on (1) one with more committed
agents
       // (2) one for which your neighbors have the most chance of satisfying
       AONTask taskJoin = pickNeighborTask(tasks);
       if(taskJoin != null){
          acceptTask(taskJoin);
          m_state = C_COMMITTED;
          m_numberOfTeamsJoined++;
          m_committedTimeRemaining = m_maxCommittedTime;
          m_lastWorkCatagory = C_JOINED;
         return;
      }//if

      float random = rand.nextFloat();
      // WaitPercent can be fixed or dynamic (based on success with current
strategy)
      Boolean wantToWait = (random < m_waitPercent);
      if(chooseToAdapt()){
         rewire();
      }else if(wantToWait){
         return;
      }else{
         //Propose a new task
         // Look at all available tasks which you can do.
         // Select the one which is best based on (1) random choice
         // (2) one which your neighbors have the most chance of satisfying
         AONTask task = pickTask(tasks);
         if(task != null){
            acceptTask(task);
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
            m_lastWorkCatagory = C_PROPOSED;
            m_numberOfTeamsJoined++;
```

```
         }//if
      }//else
   }//updateUncommitted

   /* This function returns the best task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickNeighborTask(AONTaskList tasks){
      return pickBestNeighborTask(tasks);
   }//pickNeighborTask

   /* This function returns the best task which this agent is needed for
*/
   public AONTask pickTask(AONTaskList tasks){
      return pickBestTask(tasks);
   }//pickTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "StructuralStrat_" + super.toString() + "\n";
      return value;
   }//toString
}//class StructuralStratAgent

----------------------StructuralStratAgent.java-------------------------------
-

--------------------StructuralStratImpatientAgent.java-------------------------
-

import java.util.*;

/**
 * StructuralStratImpatientAgent Class
 *   This is an agent that seeks to rewire to agents who have the most
 *   networn connections
 *   -This agent is 'strategic' because it builds off the Structural agent
 *    by choosing tasks to join and tasks to propose that are the best match
 *    for the skills possessed by the agent's uncommitted neighbors
 *   -This agent is 'impatient' because it builds off the Structural agent
 *     by choosing to drop its commitment to a task when the skills left to be
 *     filled in the task exceed the number of neighboring agents who are not
 *     committed to a task
 */
public class StructuralStratImpatientAgent extends StructuralAgent {
   /* constructor
*/
   public StructuralStratImpatientAgent(TestAON p,int id, int x, int y, Vector
agents){
      super(p,id,x,y,agents);
      m_type = "StructuralStratImpatient";
   }//constructor

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
       m_edgesChanged = 0;
       writeLog(m_id + " updateUncommitted:");
       // Look at all tasks which your immediate neighbor has committed to
       // Select the one which is best based on (1) one with more committed
agents
```

```
       // (2) one for which your neighbors have the most chance of satisfying
       AONTask taskJoin = pickNeighborTask(tasks);
       if(taskJoin != null){
          acceptTask(taskJoin);
          m_state = C_COMMITTED;
          m_numberOfTeamsJoined++;
          m_committedTimeRemaining = m_maxCommittedTime;
          m_lastWorkCatagory = C_JOINED;
         return;
      }//if

      float random = rand.nextFloat();
      // WaitPercent can be fixed or dynamic (based on success with current
strategy)
      Boolean wantToWait = (random < m_waitPercent);
      if(chooseToAdapt()){
         rewire();
      }else if(wantToWait){
         return;
      }else{
         //Propose a new task
         // Look at all available tasks which you can do.
         // Select the one which is best based on (1) random choice
         // (2) one which your neighbors have the most chance of satisfying
         AONTask task = pickTask(tasks);
         if(task != null){
            acceptTask(task);
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
            m_lastWorkCatagory = C_PROPOSED;
            m_numberOfTeamsJoined++;
         }//if
      }//else
   }//updateUncommitted

   /* This function returns the best task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickNeighborTask(AONTaskList tasks){
      return pickBestNeighborTask(tasks);
   }//pickNeighborTask

   /* This function returns the best task which this agent is needed for
*/
   public AONTask pickTask(AONTaskList tasks){
      return pickBestTask(tasks);
   }//pickTask

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      boolean dropTask = false;
      if(getUnCommittedNeighborCount() < task.unFilledSkillsCount()){
         dropTask = true;
      }//if
      return dropTask;
   }//chooseToDropTask

   /* This function returns a string representation of the agent
*/
```

```
   public String toString(){
      String value = "StructuralStratImpatient_" + super.toString() + "\n";
      return value;
   }//toString
}//class StructuralStratImpatientAgent

--------------------StructuralStratImpatientAgent.java-------------------------
-

-----------------------------TaskCluster.java---------------------------------
-

import java.util.*;
import java.awt.Point;

/**
 * TaskCluster Class
 *
 *
 */
public class TaskCluster{
   Vector<Point> m_locations;
   Vector<Integer> m_durations;
   Vector<Float> m_skillWeights;

   int m_lastTimeStep;       /*
*/
   int m_currentLocation;    /*
*/
   int m_currentDuration;    /*
*/
   int m_startTimeStep;      /*
*/
   int m_endTimeStep;        /*
*/

   int m_totalTaskCount;     /*
*/
   int m_successfulTaskCount;/*
*/
   int m_spreadCount;        /*
*/
   int m_maxSpreadCount;     /*
*/
   int m_maxTasks;           /*
*/

   /* constructor
*/
   public TaskCluster(){
      m_locations = new Vector();
      m_durations = new Vector();
      m_maxSpreadCount = 1;
      m_maxTasks = 10;
      m_startTimeStep = -1;
      m_endTimeStep = -1;
   }//TaskCluster

   /* constructor
*/
   public TaskCluster(int maxSpreadCount, int maxTasks){
      m_locations = new Vector();
      m_durations = new Vector();
```

```
      m_maxSpreadCount = maxSpreadCount;
      m_maxTasks = maxTasks;
      m_startTimeStep = -1;
      m_endTimeStep = -1;
   }//TaskCluster

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   public void addLocation(int x, int y, int duration){
      m_locations.add(new Point(x,y));
      m_durations.add(duration);
   }//addLocation

   public void addSkillWeights(Vector<Float> skillWeights){
      m_skillWeights = skillWeights;
   }//addSkillWeights

   public Point getPoint(int timeStep){
      m_currentDuration = m_currentDuration + timeStep - m_lastTimeStep;
      int duration =
((Integer)getItem(m_currentLocation,m_durations)).intValue();
      if(m_currentDuration >= duration){
         m_currentLocation++;
         m_currentDuration = 0;
         if(m_currentLocation >= getLength(m_locations)){
            m_currentLocation = getLength(m_locations)-1;
         }//if
      }//if

      m_lastTimeStep = timeStep;
      return (Point)getItem(m_currentLocation,m_locations);
```

```
   }//getPoint

   public Vector<Float> getSkillWeights(){
      return m_skillWeights;
   }//getSkillWeights

   public float getSpreadProbability(){
      float value = 0.0f;
      if(m_totalTaskCount > 0.0f){
         value = 1.0f - (float)m_successfulTaskCount / m_totalTaskCount;
      }//if
      return value;
   }//getSpreadProbability

   public int getTasksLeftCount(){
      return m_maxTasks - m_totalTaskCount;
   }//getTasksLeftCount

   public void setStartTimeStep(int startTimeStep){
      m_startTimeStep = startTimeStep;
   }//setStartTimeStep

   public void setEndTimeStep(int endTimeStep){
      m_endTimeStep = endTimeStep;
   }//setEndTimeStep

   public void addTaskCount(){
      m_totalTaskCount++;
   }//addTaskCount

   public void addSuccessCount(){
      m_successfulTaskCount++;
   }//addSuccessCount

   public TaskCluster spread(){
      if(m_spreadCount >= m_maxSpreadCount){
         return null;
      }//if
      Random rand = new Random();
      TaskCluster newCluster = null;
      if(m_totalTaskCount > 10){
         float value = 1.0f - (float)m_successfulTaskCount / m_totalTaskCount;
         if(rand.nextFloat() < value){
            newCluster = new TaskCluster();
            Point location = (Point)getItem(m_currentLocation,m_locations);
            int duration =
((Integer)getItem(m_currentLocation,m_durations)).intValue();

newCluster.addLocation((int)location.getX(),(int)location.getY(),duration);
            m_spreadCount++;
         }//if
      }//if
      return newCluster;
   }//spread

   public boolean timeStepValid(int timeStep){
      if(m_startTimeStep < 0 || m_endTimeStep < 0){
         return true;
      }//if
      if(timeStep >= m_startTimeStep && timeStep <= m_endTimeStep){
         return true;
      }//if
      return false;
```

```
   }//timeStepValid
}//TaskCluster

------------------------------TaskCluster.java-----------------------------------
-

------------------------------TaskSpike.java-------------------------------------
-

/**
 * TaskSpike Class
 *
 *
 */
public class TaskSpike{
   int m_timeStep;           /*
*/
   int m_duration;           /*
*/
   float m_taskAnnounceTime; /*
*/

   int m_savedTasksPerBatch; /*
*/
   int m_savedTaskAnnounceTime;
                             /*
*/

   /* constructor
*/
   public TaskSpike(){
      m_timeStep = 0;
      m_duration = 0;
      m_taskAnnounceTime = 0.0f;
   }//constructor

   /* constructor
*/
   public TaskSpike(int timeStep, int duration, float taskAnnounceTime){
      m_timeStep = timeStep;
      m_duration = duration;
      m_taskAnnounceTime = taskAnnounceTime;
   }//constructor

   public int getTimeStep(){
      return m_timeStep;
   }//getTimeStep

   public void setTimeStep(int timeStep){
      m_timeStep = timeStep;
   }//setTimeStep

   public int getDuration(){
      return m_duration;
   }//getDuration

   public void setDuration(int duration){
      m_duration = duration;
   }//setDuration

   public float getTaskAnnounceTime(){
      return m_taskAnnounceTime;
   }//getTaskAnnounceTime
```

```
   public void setTaskAnnounceTime(float taskAnnounceTime){
      m_taskAnnounceTime = taskAnnounceTime;
   }//setTaskAnnounceTime

   public void saveTasksPerBatch(int tasksPerBatch){
      m_savedTasksPerBatch = tasksPerBatch;
   }//saveTasksPerBatch

   public int retrieveTasksPerBatch(){
      return m_savedTasksPerBatch;
   }//retrieveTasksPerBatch

   public void saveTaskAnnounceTime(int taskAnnounceTime){
      m_savedTaskAnnounceTime = taskAnnounceTime;
   }//saveTaskAnnounceTime

   public int retrieveTaskAnnounceTime(){
      return m_savedTaskAnnounceTime;
   }//retrieveTaskAnnounceTime

   public void decrementDuration(){
      m_duration--;
   }//decrementDuration
}//class TaskSpike

-------------------------------TaskSpike.java-----------------------------------
-

-------------------------------TestAON.java-------------------------------------
-

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.IOException;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.File;
import javax.swing.Timer;
import java.awt.image.*;
import javax.imageio.ImageIO;

/**
 * TestAON Class
 *
 *
 */
public class TestAON extends Frame implements Runnable{
   /* These constants represent the possible values of an agent's state
*/
   final int C_UNCOMMITTED = 0,
                             /* The agent is not committed to a task
*/
             C_COMMITTED = 1,
                             /* The agent is committed to a task
*/
             C_ACTIVE = 2;   /* The agent is committed and working on a task
*/

   /* These constants represent the possible values of an agent's last action
*/
```

```
   final int C_WAITED = 0,   /* Last agent action was to wait
*/
             C_REWIRED = 1,  /* Last agent action was to rewire
*/
             C_PROPOSED = 2, /* Last agent action was to propose a task
*/
             C_JOINED = 3,   /* Last agent action was to join a task
*/
             C_WORKED = 4,   /* Last agent action was to work on a task
*/
             C_TRAVELLED = 5;
                             /* Last agent action was to travel
*/

   /* These constants represent the possible values of a task's state
*/
   final int C_PENDING = 0,  /* The task can still be worked on
*/
             C_SKILLS_FAIL = 1,
                             /* The task failed because of a shortage of
*/
                             /*   one or more skills
*/
             C_DISTANCE_FAIL = 2,
                             /* The task failed because one or more agents
*/
                             /*   that were committed to the task failed to
*/
                             /*   travel within range of the task before it
*/
                             /*   failed (meaning all skills were filled)
*/
             C_TASK_COMPLETE = 3;
                             /* The task was completed successfully
*/
   boolean debug;            /* This flag determines whether to write debug
*/
                             /*   information to the log file
*/
   boolean m_initializing;   /* This is true when TestAON is initializing
*/
   boolean m_drawStatusText; /* This flag determines whether to draw the
*/
                             /*   status text on the TestAON display
*/
   int m_agentCount;         /* The number of agents in the agent list
*/
   Vector<AgentX> m_agents;  /* The list of agents for the test
*/
   AONTaskList m_tasks;      /* The list of tasks for the test
*/
   int m_taskID;             /* The task id for the next task to be created
*/

   int m_columnCount;        /* The number of pixel columns in the display
*/
   int m_rowCount;           /* The number of pixel rows in the display
*/
   int m_columnOffset;       /* The number of pixel columns to offset the
*/
                             /*   display of the agents and tasks
*/
```

```
   int m_rowOffset;          /* The number of pixel rows to offset the
*/
                             /*   display of the agents and tasks
*/
   int m_ovalWidth;          /* The width of an oval representing an agent
*/
                             /*   or a task on the display
*/
   int m_maxInitialDistance; /* This distance represents the maximum distance
*/
                             /*   between agents who will have a network link
*/
                             /*   added to connect them when networking the
*/
                             /*   agents when initializing the test
*/
   int m_maximumIterations;  /* The number of timesteps to run the test
*/
   int m_iteration;          /* The current timestep for the test
*/
   int m_numberOfTeamsFormed;/* The number of teams that have been formed
*/
                             /*   which means all skills in the task the team
*/
                             /*   is attempting to complete have been filled
*/
   int m_numberOfTasksIntroduced;
                             /* The number of tasks introduced to the system
*/
   int m_numberOfOptimalTeamsFormed;
                             /* The number of teams formed in computing the
*/
                             /*   optimal performance
*/
   int m_numberOfAgentsActive;
                             /* The number of agents that have ever actively
*/
                             /*   worked on a task
*/
   int m_numberOfAgentsUpdated;
                             /* The number of agents who have been updated
*/

   int m_pendingExpiredTasks;/* The number of tasks that have expired before
*/
                             /*   being classified as a skill failire task or
*/
                             /*   a distance failure task
*/
   int m_skillFailExpiredTasks;
                             /* The number of tasks that failed because one
*/
                             /*   or more skills were not filled
*/
   int m_distanceFailExpiredTasks;
                             /* The number of tasks that failed because one
*/
                             /*   or more agents committed to the task had
*/
                             /*   not come in range of the task
*/
   int m_completeExpiredTasks;
```

```
                              /* The number of tasks completed successfully
*/

   int m_maxActiveTime;
                              /* The maximum amount of time a task can be
*/
                              /*   actively worked on
*/
   int m_maxAdvertisedTime;
                              /* The maximum amount of time a task can be
*/
                              /*   advertised in the system
*/
   int m_numberOfSkills;
                              /* The number of skills that exist in the system
*/
   int m_numberOfSkillsPerTask;
                              /* The number of skills in a task
*/
   int m_taskAnnounceTime;
                              /* The amount of between announcing a new batch
*/
                              /*   of tasks
*/

   int m_maxCommittedTime;    /* The maximum amount of time an agent can be
*/
                              /*   committed to a task
*/
   boolean firstLog;          /* This is true the first time data is written
*/
                              /*   to the log file
*/
   boolean m_autoNetworkAgents;
                              /* This flag determines whether to network the
*/
                              /*   agents after loading the agents
*/

   int m_updateInterval;      /* The number of milliseconds between updates to
*/
                              /*   the agents and tasks for interactive tests
*/
   Timer m_updateTimer;       /* The timer for when to update the agents and
*/
                              /*   tasks for an interactive test
*/
   int m_paintInterval;       /* The number of milliseconds between updates to
*/
                              /*   the display for an interactive test
*/
   Timer m_paintTimer;        /* The timer for when to update the display for
*/
                              /*   an interactive test
*/
   Vector<Integer> m_agentProcessingList;
                              /* A list of agents to be processed, created for
*/
                              /*   each time step to allow a random ordering
*/
                              /*   for updating agents
*/
   Vector<String> m_teamsFormed;
```

```
                            /* A list of team id strings for teams formed
*/
   Vector<String> m_teamsUtilityCounted;
                            /* A list of team id strings for teams whose
*/
                            /*   utility has been counted
*/
   SkillPriorityList m_agentSkillPriorityList;
                            /* A list of agents who possess each skill type
*/
                            /*   ordered by the number of agents who possess
*/
                            /*   each skill in ascending order
*/
   boolean m_firstSkillPriorityList;
                            /* This is true if the skill priority list needs
*/
                            /*   to be created (the first time it is needed)
*/
   Vector<Integer> m_tasksUsedComputeOptimal;
                            /* A list of task ids for tasks that have been
*/
                            /*   used to compute the optimal performance
*/
   String m_inputFilename;  /* The filename for the XML file containing the
*/
                            /*   test settings
*/
   String m_outputDirectory; /* The path of the directory to output results
*/
   String m_outputFilename;  /* The filename used to name output files
*/
   String m_logFilename;    /* The filename for the log file
*/
   float m_performance;     /* The percentage tasks completed successfully
*/
   float m_optimalPerformance;
                            /* The optimal percentage of tasks completed
*/
   float m_efficiency;      /* The percentage of agents who have been active
*/
   boolean m_runUnitTests;  /* This flag determines whether to run unit test
*/
   int m_runNumber;         /* The unique run id number for this test run
*/
   boolean m_exitWhenDone;  /* This flag determines whether to close TestAON
*/
                            /*   when the test has completed
*/
   int m_minAdaptTimeStep;  /* The first timestep to begin rewiring
*/
   int m_maxAdaptTimeStep;  /* The last timestep to rewire the network
*/
   String m_adaptationString;/* The name of the rewiring method agents use
*/
   int m_numberOfTasksPerBatch;
                            /* The number of tasks to create in one round of
*/
                            /*   creating new tasks
*/
   String m_description;    /* A string describing this test
*/
```

```
   boolean m_drawAgentIDs;   /* This flag determines whether to draw agent
*/
                             /*   ids next to the agent oval in the display
*/
   boolean m_drawTaskIDs;    /* This flag determines whether to draw task
*/
                             /*   ids next to the task oval in the display
*/
   boolean m_drawAON;        /* This flag determines whether to draw the GUI
*/
                             /*   for an interactive test run
*/
   int m_changedEdges;       /* The number of network connections that have
*/
                             /*   been changed
*/
   int m_averageDegree;      /* The average number of network connections per
*/
                             /*   agent
*/
   int m_averageDensity;     /* The average number of agents a network
*/
                             /*   neighborhood
*/
   int m_minDegree;          /* The fewest number of network connections an
*/
                             /*   agent has in the network
*/
   int m_maxDegree;          /* The maximum number of network connections an
*/
                             /*   agent has in the network
*/
   float m_degreeVariance;   /* The variance of the number of network
*/
                             /*   connections agents have in the network
*/
   float m_averageTaskDistance;
                             /* The average distance to a task from an agent
*/
   int m_clusterCount;       /* The number of task clusters
*/
   int m_taskCount;          /* The number of tasks created
*/

   int m_screenShotInterval; /* The number of time steps between screen shots
*/
   boolean m_saveScreenShot; /* This flag determines whether screen shots are
*/
                             /*   are automatically saved
*/
   int m_maxNodeDegree;      /* The maximum number of network connections an
*/
                             /*   agent can have
*/
   int m_communicationDepth; /* The number of network connections deep into
*/
                             /*   the network an agent can look when getting
*/
                             /*   information for rewiring
*/
   int m_neighborhoodRadius; /* The depth of network connections to look
*/
```

```
                              /*   when getting a neighborhood skill count
*/
   int m_timeWaiting;         /* The number of time steps agents have waited
*/
   int m_timeRewiring;        /* The number of time steps agents have rewired
*/
   int m_timeProposing;       /* The number of time steps agents have proposed
*/
   int m_timeJoining;         /* The number of time steps agents have joined
*/
   int m_timeWorking;         /* The number of time steps agents have worked
*/

   boolean m_running;         /* This is true when the test is still going
*/
   boolean m_writeAgentData; /* This flag determines whether to write agent
*/
                              /*   data to a CSV file for debug purposes
*/

   int m_writeDataInterval;  /* The number of time steps between writing data
*/
                              /*   to the system statistics CSV file
*/
   CSVFile m_systemDataFile; /* The system CSV file to save statistics of the
*/
                              /*   test for later analysis
*/
   boolean m_displaySpringLayout;
                              /* This flag determines whether to display the
*/
                              /*   agents using a simulated spring layout in
*/
                              /*   an interactive test run
*/
   int m_adaptationRate;     /* The rate for adapting network connections
*/

   Vector<String> m_agentTypes;
                              /* A list of agent types present in the test
*/
   Vector<Float> m_agentTypeUtility;
                              /* The utility each agent type earned
*/
   Vector<Integer> m_agentTypeCount;
                              /* The number of agents of each agent type
*/
   Hashtable m_agentTypeColor;
                              /* The color to draw each agent type with
*/
   Vector<Float> m_agentTypeRatio;
                              /* The proportion of each agent type to be
*/
                              /*   created when mixing agent types
*/
   Vector<Vector> m_agentTypeSkills;
                              /* The skills possessed by each agent type
*/
   float m_totalAONUtility;  /* The total utility achieved by the agents
*/
   int m_agentTypesChanged;  /* The number of agents who changed agent type
*/
```

```
   int m_taskRange;          /* The range an agent must be within to actively
*/
                             /*   work on a task
*/
   boolean m_useCommandSkills;
                             /* This flag determines whether to use a command
*/
                             /*   skill for each task (a skill that must be
*/
                             /*   filled before any other skills)
*/
   boolean m_useTaskRange;   /* This flag determines whether agents must be
*/
                             /*   within a task range to actively work a task
*/
   Vector<Point> m_agentCluster;
                             /* A list of points representing agent clusters
*/
   boolean m_clusterAgents;  /* This determines whether to cluster agent
*/
                             /*   locations
*/
   Vector<TaskCluster> m_taskCluster;
                             /* A list of task clusters
*/
   Vector<TaskSpike> m_taskSpikes;
                             /* A list of task spikes where the number of
*/
                             /*   tasks created per batch of tasks increases
*/
   boolean m_clusterTasks;   /* This determines whether to cluster tasks
*/
   boolean m_useNodeFailure; /* This determines whether to use node failure
*/
   Vector<NodeFailure> m_nodeFailure;
                             /* A list of node failures where agents fail at
*/
                             /*   a specified time step
*/
   Vector<Integer> m_failedNodes;
                             /* A list of failed nodes or agent ids in the
*/
                             /*   network
*/
   boolean m_createAgentsOnly;
                             /* This determines whether to create agents then
*/
                             /*   exit TestAON
*/
   boolean m_clusterSpread;  /* This determines whether to spread clusters
*/
   int m_maxClusterSpread;   /* This is the maximum number of times a cluster
*/
                             /*   can spread to create new clusters
*/
   int m_maxClusterTasks;    /* This is the maximum number of tasks a cluster
*/
                             /*   can produce
*/
   boolean m_useMixedAgents; /* This determines whether to use mixed agents
*/

   MyMessageFrame m_messageFrame;
```

```
                              /* This displays agent and task information when
*/
                              /*   running an interactive test
*/
   AgentX m_selectedAgent;    /* The currently selected agent that has its
*/
                              /*   information displayed in the MessageFrame
*/
   AONTask m_selectedTask;    /* The currently selected task that has its
*/
   Random rand;               /*   information displayed in the MessageFrame
*/
   boolean m_useGlobalRandomSeed;
                              /* This determines whether to use one random
*/
                              /*   for all tasks and agents
*/
   int m_randomSeed;          /* The random seed for all tasks and agents
*/
   Roads m_roads;             /* A set of roads agents can traverse
*/
   boolean m_useRoads;        /* This flag determines whether to use roads
*/
   Hashtable m_agentSkillHistogram;
                              /* A count of the number of agents who possess
*/
                              /*   each skill
*/
   int m_distanceTravelled;   /* The distance agents have travelled
*/
   int m_impossibleTaskCount;/* The number of tasks that are impossible given
*/
                              /*   the skills required and the skills
available*/
   Vector<Integer> m_unfilledSkillsCount;
                              /* The number of times each skill has gone
*/
                              /*   unfilled in a task
*/
   Vector<Integer> m_skillDemandCount;
                              /* The number of times each skill has been
*/
                              /*   required in a task
*/
   int m_joinedImpossibleTaskCount;
                              /* The number of times agents have joined an
*/
                              /*   impossible task
*/
   int m_proposedImpossibleTaskCount;
                              /* The number of times agents have proposed an
*/
                              /*   impossible task
*/
   int m_repeatTaskCount;     /* The number of times agents have committed to
*/
                              /*   to a task they were previously committed to
*/
   int m_neighborhoodSkillShortageSum;
                              /* The number of skills not available in agent's
*/
                              /*   neighborhoods
*/
```

```
   int m_minNeighborhoodSkillShortage;
                              /* The smallest number of skills missing in an
*/
                              /*   agent's neighborhood
*/
   int m_maxNeighborhoodSkillShortage;
                              /* The largets number of skills missing in an
*/
                              /*   agent's neighborhood
*/
   float m_averageNeighborhoodSkillShortage;
                              /* The average number of skills missing in
*/
                              /*   network neighborhoods
*/
   int m_aliveAgentCount;     /* The number of agents who have not failed
*/
   int m_skillHistoryLength; /* The number of time steps to maintain a
*/
                              /*   history of skills that have caused tasks to
*/
                              /*   fail
*/

   /* constructor
*/
   public TestAON(){
      debug = false;
      m_drawStatusText = true;

      addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e){
            dispose();
            System.exit(0);
         }//windowClosing
      });//addWindowListener

      ActionListener timerListener = new ActionListener(){
         public void actionPerformed(ActionEvent ae){
            runTimeStep();
         }//actionPerformed
      };//timerListener

      ActionListener paintListener = new ActionListener(){
         public void actionPerformed(ActionEvent ae){
            repaint();
            updateMessageFrame();
         }//actionPerformed
      };//paintListener

      addKeyListener(new KeyAdapter(){
         public void keyPressed(KeyEvent ke){
            switch(ke.getKeyCode()){
               case(KeyEvent.VK_S):
                  m_displaySpringLayout = !m_displaySpringLayout;
                  if(!m_paintTimer.isRunning()){
                      repaint();
                  }//if
                  break;
               default:
            }//switch
         }//keyPressed
      });//addKeyListener
```

```
        m_initializing = false;

        m_agentCount = 100; //N
        m_agents = new Vector();
        m_tasks = new AONTaskList();
        m_taskID = 0;

        m_columnCount = 700;
        m_rowCount = 700;
        m_columnOffset = 20;
        m_rowOffset = 50;
        m_ovalWidth = 10;
        firstLog =  true;
        m_autoNetworkAgents = true;

        AONMouseInputListener mouseListener = new
AONMouseInputListener(m_columnOffset,m_rowOffset,m_agents,m_tasks,this);

        addMouseListener(mouseListener);
        addMouseMotionListener(mouseListener);

        m_maxInitialDistance = 50;
        m_maximumIterations = 100;
        m_iteration = 0;
        m_numberOfTeamsFormed = 0;
        m_numberOfTasksIntroduced = 0;
        m_numberOfAgentsActive = 0;
        m_numberOfAgentsUpdated = 0;

        m_maxActiveTime = 10;
        m_maxAdvertisedTime = 10;
        m_numberOfSkills = 10;
        m_numberOfSkillsPerTask = 3;
        m_taskAnnounceTime = 2;

        m_maxCommittedTime = 10;

        m_updateInterval = 500;
        m_updateTimer = new Timer(m_updateInterval, timerListener);
        m_paintInterval = 1000;
        m_paintTimer = new Timer(m_paintInterval, paintListener);
        m_agentProcessingList = new Vector();
        m_teamsFormed = new Vector();
        m_teamsUtilityCounted = new Vector();
        m_agentSkillPriorityList = new SkillPriorityList();
        m_firstSkillPriorityList = true;
        m_tasksUsedComputeOptimal = new Vector();

        m_inputFilename = "AON_Settings.xml";
        m_outputDirectory = "";
        m_outputFilename = "AONResults";
        m_logFilename = "AONLog";
        m_performance = 0.0f;
        m_efficiency = 0.0f;
        m_runUnitTests = false;
        m_runNumber = 1;
        m_exitWhenDone = false;
        m_minAdaptTimeStep = 0;
        m_maxAdaptTimeStep = 0;
        m_adaptationString = "AgentX";
        m_numberOfTasksPerBatch = 100;
        m_description = "";
```

```
m_drawAgentIDs = false;
m_drawAON = true;
m_changedEdges = 0;
m_averageDegree = 0;
m_averageDensity = 0;
m_minDegree = 0;
m_maxDegree = 0;
m_degreeVariance = 0.0f;

m_screenShotInterval = 1;
m_saveScreenShot = false;
m_maxNodeDegree = 1000;
m_communicationDepth = 2;
m_neighborhoodRadius = 10;
m_timeWaiting = 0;
m_timeRewiring = 0;
m_timeProposing = 0;
m_timeJoining = 0;
m_timeWorking = 0;

m_running = true;
m_writeAgentData = false;

m_writeDataInterval = 10;
m_systemDataFile = new CSVFile();
m_displaySpringLayout = false;

m_adaptationRate = 1;
m_agentTypes = new Vector();
m_agentTypeUtility = new Vector();
m_agentTypeCount = new Vector();
m_agentTypeColor = new Hashtable();
m_agentTypeRatio = new Vector();
m_agentTypeSkills = new Vector();

m_totalAONUtility = 0.0f;
m_useCommandSkills = false;
m_useTaskRange = false;
m_agentCluster = new Vector();
m_clusterAgents = false;
m_taskCluster = new Vector();
m_taskSpikes = new Vector();
m_clusterTasks = false;
m_useNodeFailure = false;
m_nodeFailure = new Vector();
m_failedNodes = new Vector();
m_taskRange = 100;
m_createAgentsOnly = false;
m_clusterSpread = false;
m_maxClusterSpread = 1;
m_maxClusterTasks = 10;
m_useMixedAgents = true;

m_messageFrame = new MyMessageFrame("Agent Data",600,200,0,null);
rand = new Random(System.currentTimeMillis());
m_useGlobalRandomSeed = false;
m_roads = new Roads();
m_useRoads = false;
m_agentSkillHistogram = new Hashtable();
m_unfilledSkillsCount = new Vector();
for(int i = 0; i < m_numberOfSkills; i++){
   m_unfilledSkillsCount.add(0);
}//for
```

```
      m_skillDemandCount = new Vector();
      for(int i = 0; i < m_numberOfSkills; i++){
         m_skillDemandCount.add(0);
      }//for

      m_aliveAgentCount = 0;
      m_skillHistoryLength = -1;
   }//constructor

   /* This function sets the name of the method used by the agents to rewire
*/
   /*   their network connections
*/
   public void setAdaptation(String adaptationName){
      m_adaptationString = adaptationName;
   }//setAdaptation

   /* This function sets the path of the directory to place the output of the
*/
   /*   test in
*/
   public void setOutputDirectory(String outputDirectory){
      m_outputDirectory = outputDirectory;

      File directoryFile = new File(outputDirectory);
      if(!directoryFile.exists()){
         directoryFile.mkdir();
      }//if
   }//setOutputDirectory

   /* This function sets the description string for the test being run
*/
   public void setDescription(String description){
      m_description = description;
   }//setDescription

   /* This function sets the maximum number of network connections one agent
*/
   /*   can have to its neighbors
*/
   public void setMaxNodeDegree(int maxNodeDegree){
      m_maxNodeDegree = maxNodeDegree;
   }//setMaxNodeDegree

   /* This function sets the depth in the local network an agent can look
*/
   /*   when gathering information to rewire
*/
   public void setCommunicationDepth(int communicationDepth){
      m_communicationDepth = communicationDepth;
   }//setCommunicationDepth

   /* This function sets the depth in the local network an agent can look
*/
   /*   when counting neighborhood skills
*/
   public void setNeighborhoodRadius(int neighborhoodRadius){
      m_neighborhoodRadius = neighborhoodRadius;
   }//setNeighborhoodRadius

   /* This function sets whether to use skill 0 as a command skill, or a
*/
```

```
   /*   skill that must be filled before the other skills can be filled
*/
   public void setUseCommandSkills(boolean useCommandSkills){
      System.out.println("SET: UseCommandSkills=" + useCommandSkills);
      m_useCommandSkills = useCommandSkills;
   }//setUseCommandSkills

   /* This function sets the distance in pixels from a task an agent must be
*/
   /*   to actively work on the task
*/
   public void setTaskRange(int taskRange){
      m_useTaskRange = true;
      m_taskRange = taskRange;
   }//setTaskRange

   /* This function sets the filename to be used to get input settings for
*/
   /*   running the test
*/
   public void setInputFilename(String inputFilename){
      m_inputFilename = inputFilename;
   }//setInputFilename

   /* This function sets a unique identifier for this test run that
*/
   /*   represents the number of times the test has been repeated
*/
   public void setRunNumber(int runNumber){
      m_runNumber = runNumber;
   }//setRunNumber

   /* This function returns the maximum number of skills in the system
*/
   public int getSkillMax(){
      return  m_numberOfSkills;
   }//getSkillMax

   /* This function returns whether the network is being displayed with
*/
   /*   network connections that have simulated spring characteristics
*/
   public boolean getDisplaySpringLayout(){
      return m_displaySpringLayout;
   }//getDisplaySpringLayout

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
```

```
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getLength

   /* This function is the entry point for TestAON when it is called from the
*/
   /*   command line
*/
   public static void main(String args[]){
      System.out.println("TestAON:");
      TestAON mainFrame = new TestAON();

      for(int i = 0; i < args.length; i++){
         mainFrame.processArg(i,args);
      }//for

      mainFrame.run();
   }//main

   /* This function is called from TestAON's main function when run from the
*/
   /*   command line, or is the entry point to TestAON when run as a Runnable
*/
   /*   object from another class (RunTestAON, for example)
*/
   public void run(){
      if(m_inputFilename != ""){
         readInputFile();
      }//if

      System.out.println("Run " + m_runNumber
                         + " \nN (agentCount)=" + m_agentCount + " alpha
(activeTime)=" + m_maxActiveTime
                         + " \ngamma (advertisedTime)=" + m_maxAdvertisedTime
+ " sigma (Skills)=" + m_numberOfSkills
                         + " \n|T|=" + m_numberOfSkillsPerTask + " mu
(announceTime)=" + m_taskAnnounceTime
                         + " \nMaxNodeDegree=" + m_maxNodeDegree + "
CommunicationDepth=" + m_communicationDepth
                         + " \nTaskRange=" + m_taskRange + "
NeighborhoodRadius=" + m_neighborhoodRadius);

      setSize(m_columnCount,m_rowCount);
      setTitle("TestAON");
      if(m_drawAON){
         setVisible(true);
      }//if

      if(m_runUnitTests){
         System.out.println("runUnitTest...");
```

```
         printStructuralAgents();
         printPerformanceAgents();
         printInventoryAgents();
         printEgalitarianAgents();
         printCommunicationDepth();
         System.exit(0);
      }else if(m_createAgentsOnly){
         //Create agents
         System.out.println(m_agentTypes);
         m_agents.clear();
         createAgents();
         writeAgents();
         System.out.println("Agent count= " + getLength(m_agents));
         System.exit(0);
      }else{
         initializeAON();
         startTimer();
      }//else
   }//run

   /* This function processes command line arguments for TestAON
*/
   private boolean processArg(int i, String args[]){
      try{
         if(args[i].compareTo("-test") == 0){
            m_runUnitTests = true;
         }else if(args[i].compareTo("-exit") == 0){
            m_exitWhenDone = true;
         }else if(args[i].compareTo("-dist") == 0){
            m_maxInitialDistance = Integer.parseInt(args[i+1]);
         }else if(args[i].compareTo("-iter") == 0){
            m_maximumIterations = Integer.parseInt(args[i+1]);
         }else if(args[i].compareTo("-run_id") == 0){
            m_runNumber = Integer.parseInt(args[i+1]);
         }else if(args[i].compareTo("-min_adapt_step") == 0){
            m_minAdaptTimeStep = Integer.parseInt(args[i+1]);
         }else if(args[i].compareTo("-paint_time") == 0){
            m_paintInterval = Integer.parseInt(args[i+1]);
         }else if(args[i].compareTo("-update_time") == 0){
            m_updateInterval = Integer.parseInt(args[i+1]);
         }else if(args[i].compareTo("-description") == 0){
            m_description = args[i+1];
         }else if(args[i].compareTo("-input") == 0){
            m_inputFilename = args[i+1];
         }else if(args[i].compareTo("-output_directory") == 0){
            m_outputDirectory = args[i+1];
         }else if(args[i].compareTo("-create") == 0){
            m_createAgentsOnly = true;
         }//else if
      }catch(Exception ex){
         System.out.println("Invalid argument");
         System.out.println("Usage: TestAON -dist <number>[maximum initial
distance]");
         System.out.println("-iter <number>[number of iterations]");
         System.out.println("-test [run the test cases]");
         System.out.println("-run_id <number>[the run number]");
         System.out.println("-exit [exit when run has been completed]");
         System.out.println("-min_adapt_step <number>[the minimum time step to
start adapting network]");
         System.out.println("-performance_adapt [use performance based network
adaptation]");
         System.out.println("-structural_adapt [use structural based network
adaptation]");
```

```
         System.out.println("-paint_time <number>[the number of milliseconds
between calls to the paint function]");
         System.out.println("-update_time <number>[the number of milliseconds
between calls to the update function]");
         System.out.println("-description <text>[a description for the test
run]");
         System.out.println("-input <text>[the filename of the xml file
containing the AON input parameters]");
         System.out.println("-output_directory <text>[the path of the directory
to use as the location to write the output files]");
         return false;
      }//catch
      return true;
   }//processArg

   /* This function reads an XML input file, getting test settings for the
*/
   /*   test to be run
*/
   private void readInputFile(){
      System.out.println("readInputFile");
      AONXMLReader xmlFile = new AONXMLReader();
      if(!xmlFile.openXML(m_inputFilename)){
       System.out.println("Bad XML File: " + m_inputFilename + " Using default
AON settings");
       return;
      }//if

      XMLElement root = xmlFile.getRoot();
      String attributeName = "";
      String value = "";
      try{
         attributeName = "RandomSeed";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null){
            m_randomSeed = Integer.parseInt(value);
            m_useGlobalRandomSeed = true;
            rand = new Random(m_randomSeed);
         }//if

         attributeName = "AgentCount";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_agentCount =
Integer.parseInt(value);

         attributeName = "MaxInitialDistance";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_maxInitialDistance =
Integer.parseInt(value);

         attributeName = "MaxIterations";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_maximumIterations =
Integer.parseInt(value);

         attributeName = "MaxActiveTime";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_maxActiveTime =
Integer.parseInt(value);

         attributeName = "MaxAdvertisedTime";
         value = root.getAttribute(attributeName);
```

```
         if(value != "" && value != null)m_maxAdvertisedTime =
Integer.parseInt(value);

         attributeName = "NumberOfSkills";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_numberOfSkills =
Integer.parseInt(value);

         attributeName = "SkillsPerTask";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_numberOfSkillsPerTask =
Integer.parseInt(value);

         float floatValue = 0.0f;
         attributeName = "TaskAnnounceTime";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)floatValue = Float.parseFloat(value);
         processTaskAnnounceTime(floatValue);

         attributeName = "MaxCommittedTime";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_maxCommittedTime =
Integer.parseInt(value);

         attributeName = "UpdateInterval";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_updateInterval =
Integer.parseInt(value);

         attributeName = "PaintInterval";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_paintInterval =
Integer.parseInt(value);

         attributeName = "OutputFilename";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_outputFilename = value;

         attributeName = "MinAdaptTimeStep";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_minAdaptTimeStep =
Integer.parseInt(value);

         attributeName = "MaxAdaptTimeStep";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null){
            m_maxAdaptTimeStep = Integer.parseInt(value);
         }else{
            m_maxAdaptTimeStep = m_maximumIterations;
         }//else

         attributeName = "DrawAON";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_drawAON = parseBoolean(value);

         attributeName = "DrawAgentIDs";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_drawAgentIDs = parseBoolean(value);

         attributeName = "DrawTaskIDs";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_drawTaskIDs = parseBoolean(value);
```

```
         attributeName = "ScreenShotInterval";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_screenShotInterval =
Integer.parseInt(value);

         attributeName = "SaveScreenShot";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_saveScreenShot =
parseBoolean(value);

         attributeName = "MaxNodeDegree";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_maxNodeDegree =
Integer.parseInt(value);

         attributeName = "CommunicationDepth";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_communicationDepth =
Integer.parseInt(value);

         attributeName = "NeighborhoodRadius";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_neighborhoodRadius =
Integer.parseInt(value);

         attributeName = "WriteDataInterval";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_writeDataInterval =
Integer.parseInt(value);

         attributeName = "WriteAgentData";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_writeAgentData =
parseBoolean(value);

         attributeName = "AdaptationRate";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_adaptationRate =
Integer.parseInt(value);

         attributeName = "UseCommandSkills";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_useCommandSkills =
parseBoolean(value);

         attributeName = "TaskRange";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null){
            m_useTaskRange = true;
            m_taskRange = Integer.parseInt(value);
         }//if

         attributeName = "SkillHistoryLength";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null)m_skillHistoryLength =
Integer.parseInt(value);

         attributeName = "RoadsFile";
         value = root.getAttribute(attributeName);
         if(value != "" && value != null){
            if(m_roads.open(value)){
               m_useRoads = true;
            }//if
```

```
         }//if

         if(m_adaptationString.compareTo("AgentX") == 0){
            System.out.println("Using first test settings...");
            //Get first Test in xml settings file
            XMLElement xmlTests = root.getElement("Tests");
            XMLElement xmlTest = xmlTests.getElement(0);
            attributeName = "Adaptation";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null){
               m_adaptationString = value;
               m_useMixedAgents = false;
            }//if

            attributeName = "OutputDirectory";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)m_outputDirectory = value;

            attributeName = "Description";
            value = xmlTest.getAttribute(attributeName);
            if(value != "" && value != null)m_description = value;
         }//if

         System.out.println("Read AON parameters from " + m_inputFilename);
      }catch(Exception ex){
         System.out.print("Error loading attribute " + attributeName + " from
xml file: ");
         ex.printStackTrace();
      }//catch

      try{
         XMLElement xmlAgentTypes = root.getElement("AgentTypes");
         if(xmlAgentTypes != null){
            int agentTypesCount = xmlAgentTypes.getNumElements();
            for(int i = 0; i < agentTypesCount; i++){
               XMLElement xmlAgentType = xmlAgentTypes.getElement(i);
               String type = xmlAgentType.getAttribute("Name");
               String colorName = xmlAgentType.getAttribute("Color");
               float ratio =
Float.parseFloat(xmlAgentType.getAttribute("Ratio"));
               m_agentTypes.add(type);
               m_agentTypeUtility.add(0.0f);
               m_agentTypeCount.add(0);
               m_agentTypeColor.put(type,colorName);
               m_agentTypeRatio.add(ratio);
               Vector<Integer> agentSkills = new Vector();
               XMLElement xmlSkills = xmlAgentType.getElement("Skills");
               if(xmlSkills != null){
                  for(int j = 0; j < xmlSkills.getNumElements(); j++){
                     XMLElement xmlSkill = xmlSkills.getElement(j);

agentSkills.add(Integer.parseInt(xmlSkill.getAttribute("SkillID")));
                  }//for
               }//if
               if(getLength(agentSkills) == 0){
                  agentSkills.add(-1);
               }//if
               m_agentTypeSkills.add(agentSkills);
            }//for
            System.out.println("Read Agent Types from " + m_inputFilename);
         }//if
      }catch(Exception ex){
         System.out.println("Error reading Agent Types from xml file: " + ex);
```

```
         ex.printStackTrace();
      }//catch

      try{
         XMLElement xmlAgentClusters = root.getElement("AgentClusters");
         if(xmlAgentClusters != null){
            m_clusterAgents = true;
            int agentClustersCount = xmlAgentClusters.getNumElements();
            for(int i = 0; i < agentClustersCount; i++){
               XMLElement xmlAgentType = xmlAgentClusters.getElement(i);
               int x = Integer.parseInt(xmlAgentType.getAttribute("X"));
               int y = Integer.parseInt(xmlAgentType.getAttribute("Y"));
               Point cluster = new Point(x,y);
               m_agentCluster.add(cluster);
            }//for
            System.out.println("Read Agent Clusters from " + m_inputFilename);
         }//if
      }catch(Exception ex){
         System.out.println("Error reading Agent Clusters from xml file: " +
ex);
      }//catch

      try{
         XMLElement xmlTaskClusters = root.getElement("TaskClusters");
         if(xmlTaskClusters != null){
            m_clusterTasks = true;
            int TaskClustersCount = xmlTaskClusters.getNumElements();
            for(int i = 0; i < TaskClustersCount; i++){
               XMLElement xmlTaskCluster = xmlTaskClusters.getElement(i);
               if(xmlTaskCluster.getAttribute("MaxTaskCount") != null){
                  m_maxClusterTasks =
Integer.parseInt(xmlTaskCluster.getAttribute("MaxTaskCount"));
               }//if
               TaskCluster cluster = new
TaskCluster(m_maxClusterSpread,m_maxClusterTasks);
               if(xmlTaskCluster.getAttribute("StartTime") != null){

cluster.setStartTimeStep(Integer.parseInt(xmlTaskCluster.getAttribute("StartTim
e")));
               }//if
               if(xmlTaskCluster.getAttribute("EndTime") != null){

cluster.setEndTimeStep(Integer.parseInt(xmlTaskCluster.getAttribute("EndTime"))
);
               }//if
               XMLElement xmlClusterLocation =
xmlTaskCluster.getElement("Locations");
               for(int j = 0; j < xmlClusterLocation.getNumElements(); j++){
                  XMLElement location = xmlClusterLocation.getElement(j);
                  int x = Integer.parseInt(location.getAttribute("X"));
                  int y = Integer.parseInt(location.getAttribute("Y"));
                  int duration =
Integer.parseInt(location.getAttribute("Duration"));
                  System.out.println("x=" + x + " y=" + y + " duration=" +
duration);
                  cluster.addLocation(x,y,duration);
               }//for
               Vector<Float> skillWeights = new Vector();
               XMLElement xmlClusterSkillWeights =
xmlTaskCluster.getElement("SkillWeights");
               for(int j = 0; j < xmlClusterSkillWeights.getNumElements();
j++){
```

```
                  XMLElement xmlSkillWeight =
xmlClusterSkillWeights.getElement(j);
                  float weight =
Float.parseFloat(xmlSkillWeight.getAttribute("Weight"));
                  skillWeights.add(weight);
               }//for
               cluster.addSkillWeights(skillWeights);
               m_taskCluster.add(cluster);
            }//for
            m_clusterCount = getLength(m_taskCluster);
            System.out.println("Read Task Clusters from " + m_inputFilename);
         }//if
      }catch(Exception ex){
         System.out.println("Error reading Task Clusters from xml file: ");
         ex.printStackTrace();
      }//catch

      try{
         XMLElement xmlTaskSpikes = root.getElement("TaskSpikes");
         if(xmlTaskSpikes != null){
            String inputSpikesFilename = "";
            attributeName = "Filename";
            value = xmlTaskSpikes.getAttribute(attributeName);
            if(value != "" && value != null)inputSpikesFilename = value;
            System.out.println(inputSpikesFilename);

            if(inputSpikesFilename != ""){
               AONXMLReader xmlTaskSpikesFile = new AONXMLReader();
               if(xmlTaskSpikesFile.openXML(inputSpikesFilename)){
                  XMLElement xmlTaskSpikesRoot = xmlTaskSpikesFile.getRoot();
                  if(xmlTaskSpikesRoot.getElement("TaskSpikes") != null){
                     xmlTaskSpikes =
xmlTaskSpikesRoot.getElement("TaskSpikes");
                  }//if
               }//if
            }//if

            int taskSpikesCount = xmlTaskSpikes.getNumElements();
            for(int i = 0; i < taskSpikesCount; i++){
               XMLElement xmlTaskSpike = xmlTaskSpikes.getElement(i);
               int timeStep =
Integer.parseInt(xmlTaskSpike.getAttribute("TimeStep"));
               int duration =
Integer.parseInt(xmlTaskSpike.getAttribute("Duration"));
               float taskAnnounceTime =
Float.parseFloat(xmlTaskSpike.getAttribute("TaskAnnounceTime"));
               TaskSpike taskSpike = new
TaskSpike(timeStep,duration,taskAnnounceTime);
               m_taskSpikes.add(taskSpike);
            }//for
         }//if
      }catch(Exception ex){
         System.out.println("Error reading Task Spikes from xml file: ");
         ex.printStackTrace();
      }//catch

      try{
         XMLElement xmlNodeFailures = root.getElement("NodeFailures");
         if(xmlNodeFailures != null){
            String inputFailuresFilename = "";
            attributeName = "Filename";
            value = xmlNodeFailures.getAttribute(attributeName);
            if(value != "" && value != null)inputFailuresFilename = value;
```

```
            System.out.println(inputFailuresFilename);

            if(inputFailuresFilename != ""){
               AONXMLReader xmlFailuresFile = new AONXMLReader();
               if(xmlFailuresFile.openXML(inputFailuresFilename)){
                  XMLElement xmlFailuresRoot = xmlFailuresFile.getRoot();
                  if(xmlFailuresRoot.getElement("NodeFailures") != null){
                     xmlNodeFailures =
xmlFailuresRoot.getElement("NodeFailures");
                  }//if
               }//if
            }//if

            m_useNodeFailure = true;
            int NodeFailuresCount = xmlNodeFailures.getNumElements();
            for(int i = 0; i < NodeFailuresCount; i++){
               XMLElement xmlNodeFailure = xmlNodeFailures.getElement(i);
               int agentID = -1;
               int timeStep =
Integer.parseInt(xmlNodeFailure.getAttribute("TimeStep"));
               int regenerateTime = -1;
               int percent = 0;
               value = xmlNodeFailure.getAttribute("RegenerationTime");
               if(value != null && value != "")regenerateTime =
Integer.parseInt(value);
               value = xmlNodeFailure.getAttribute("Percent");
               if(value != null && value != "")percent =
Integer.parseInt(value);

               if(percent > 0){
                  int agentFailureCount = m_agentCount*percent/100;
                  Vector<Integer> agentFailures = new Vector();
                  for(int j = 0; j < agentFailureCount; j++){
                     agentID = rand.nextInt(m_agentCount);
                     while(agentFailures.contains(agentID)){
                        agentID = rand.nextInt(m_agentCount);
                     }//while
                     agentFailures.add(agentID);
                     NodeFailure failures = new
NodeFailure(agentID,timeStep,regenerateTime);
                     m_nodeFailure.add(failures);
                  }//for
               }else{
                  int agentFailureCount = xmlNodeFailure.getNumElements();
                  for(int j = 0; j < agentFailureCount; j++){
                     XMLElement agentFailure = xmlNodeFailure.getElement(j);
                     agentID =
Integer.parseInt(agentFailure.getAttribute("AgentID"));
                     NodeFailure failures = new
NodeFailure(agentID,timeStep,regenerateTime);
                     m_nodeFailure.add(failures);
                  }//for
               }//else
            }//for
            System.out.println("Read Node Failures from " + m_inputFilename);
         }//if
      }catch(Exception ex){
         System.out.println("Error reading Node Failures from xml file: ");
         ex.printStackTrace();
      }//catch
      writeLog("Number of node failures: " + getLength(m_nodeFailure));

      if(!m_createAgentsOnly){
```

```
         try{
            XMLElement xmlAgents = root.getElement("Agents");
            if(xmlAgents != null){
               String inputAgentsFilename = "";
               attributeName = "Filename";
               value = xmlAgents.getAttribute(attributeName);
               if(value != "" && value != null)inputAgentsFilename = value;

               if(inputAgentsFilename != ""){
                  AONXMLReader xmlAgentsFile = new AONXMLReader();
                  if(xmlAgentsFile.openXML(inputAgentsFilename)){
                     XMLElement xmlAgentsRoot = xmlAgentsFile.getRoot();
                     if(xmlAgentsRoot.getElement("Agents") != null){
                        System.out.println("Loaded agent file " +
inputAgentsFilename);
                        xmlAgents = xmlAgentsRoot.getElement("Agents");
                     }//if
                  }//if
               }//if

               int agentCount = xmlAgents.getNumElements();
               for(int i = 0; i < agentCount; i++){
                  XMLElement xmlAgent = xmlAgents.getElement(i);
                  int id = Integer.parseInt(xmlAgent.getAttribute("ID"));
                  int x = Integer.parseInt(xmlAgent.getAttribute("X"));
                  int y = Integer.parseInt(xmlAgent.getAttribute("Y"));
                  String type = m_adaptationString;
                  if(m_useMixedAgents){
                     String temp = xmlAgent.getAttribute("Type");
                     if(temp != null && temp != ""){
                        type = temp;
                     }//if
                  }//if
                  if(!contains(type,m_agentTypes)){
                     m_agentTypes.add(type);
                     m_agentTypeCount.add(1);
                  }else{
                     int agentTypeIndex = m_agentTypes.indexOf(type);
                     int count =
((Integer)getItem(agentTypeIndex,m_agentTypeCount)).intValue();
                     setItem(agentTypeIndex,m_agentTypeCount,count+1);
                  }//else

                  AgentX agent = addAgent(id,x,y,type);
                  initializeAgent(agent);
                  XMLElement xmlSkills = xmlAgent.getElement("Skills");
                  int skillCount = xmlSkills.getNumElements();
                  for(int j = 0; j < skillCount; j++){
                     XMLElement xmlSkill = xmlSkills.getElement(j);
                     int skill =
Integer.parseInt(xmlSkill.getAttribute("SkillID"));
                     agent.addSkill(skill);
                  }//for
                  agent.setAdaptNetwork(false);

                  attributeName = "AdaptationRate";
                  value = xmlAgent.getAttribute(attributeName);
                  if(value != "" && value != null){
                     int adaptationRate = Integer.parseInt(value);
                     agent.setAdaptationRate(adaptationRate);
                  }else{
                     agent.setAdaptationRate(m_adaptationRate);
                  }//else
```

```
                attributeName = "ShapeID";
                value = xmlAgent.getAttribute(attributeName);
                if(value != "" && value !=
null)agent.setShapeID(Integer.parseInt(value));
             }//for
          }//if
          System.out.println("Read Agents from " + m_inputFilename);
       }catch(Exception ex){
          System.out.print("Error reading Agents from xml file: ");
          ex.printStackTrace();
       }//catch
    }//if
 }//readInputFile

 /* This function takes a string representation of a boolean and
*/
 /*   returns its boolean value
*/
 private boolean parseBoolean(String value){
    if(value.compareToIgnoreCase("true") == 0){
      return true;
    }else{
      return false;
    }//else
 }//parseBoolean

 /* This function takes the task announce time and assigns the proper value
*/
 /*   to the number of tasks per batch of creating new tasks
*/
 private void processTaskAnnounceTime(float taskAnnounceTime){
    if(taskAnnounceTime < 1.0f){
       m_numberOfTasksPerBatch = (int)(1.0f/taskAnnounceTime);
       m_taskAnnounceTime = 1;
    }else{
       m_numberOfTasksPerBatch = 1;
       m_taskAnnounceTime = (int)taskAnnounceTime;
    }//else
 }//processTaskAnnounceTime

 /* This function takes a string name for a color and returns the
*/
 /*   corresponding Color object
*/
 private Color getColor(String colorName){
    if(colorName == null){
       colorName = "lightGray";
    }//if
    if(colorName.compareTo("black") == 0){
       return Color.black;
    }else if(colorName.compareTo("blue") == 0){
       return Color.blue;
    }else if(colorName.compareTo("cyan") == 0){
       return Color.cyan;
    }else if(colorName.compareTo("darkGray") == 0){
       return Color.darkGray;
    }else if(colorName.compareTo("gray") == 0){
       return Color.gray;
    }else if(colorName.compareTo("green") == 0){
       return Color.green;
    }else if(colorName.compareTo("lightGray") == 0){
       return Color.lightGray;
```

```
      }else if(colorName.compareTo("magenta") == 0){
         return Color.magenta;
      }else if(colorName.compareTo("orange") == 0){
         return Color.orange;
      }else if(colorName.compareTo("pink") == 0){
         return Color.pink;
      }else if(colorName.compareTo("red") == 0){
         return Color.red;
      }else if(colorName.compareTo("white") == 0){
         return Color.white;
      }else if(colorName.compareTo("yellow") == 0){
         return Color.yellow;
      }else{
         int r = rand.nextInt(256), g = rand.nextInt(256), b =
rand.nextInt(256);
         return new Color(r,g,b);
      }//else
   }//getColor

   /* This function sets the agent that is the currently selected agent so
*/
   /*   that this agent's information can be displayed in the MessageFrame
*/
   public void setSelectedAgent(AgentX agent){
      m_selectedAgent = agent;
   }//setSelectedAgent

   /* This function sets the task that is the currently selected task so
*/
   /*   the task's information can be displayed in the MessageFrame
*/
   public void setSelectedTask(AONTask task){
      m_selectedTask = task;
   }//setSelectedTask

   /* This function starts the test executing for TestAON, either using a
*/
   /*   timer to update the next time step and draw the agents for an
*/
   /*   interactive test, or running the test with no display as fast as the
*/
   /*   computer can
*/
   private void startTimer(){
      System.out.println("Start...");
      if(m_drawAON){
         m_updateTimer.setDelay(m_updateInterval);
         m_updateTimer.start();
         m_paintTimer.setDelay(m_paintInterval);
         m_paintTimer.start();
      }else{
         m_exitWhenDone = true;
         while(m_running){
            runTimeStep();
         }//while
         printAgents();
      }//else
   }//startTimer

   /* This function writes the agents to the message log
*/
   public void printAgents(){
      int totalEdge=0;
```

```
      for(int i = 0; i < getLength(m_agents); i++){
         AgentX agent = (AgentX)getItem(i,m_agents);
         writeLog(agent.toString());
         totalEdge+=agent.getDegree();
      }//for
      writeLog("total edges " + totalEdge);
   }//printAgents

   /* This function generates some initial tasks before the test has begun
*/
   public void generateInitialTasks(){
      int startTasks = m_agentCount/m_numberOfSkillsPerTask;
      for (int i = 0; i < startTasks; i++){
         generateTask(m_maxAdvertisedTime +1);
      }//for
   }//generateInitialTasks

   /* This function draws the agents and the tasks in the GUI for an
*/
   /*   interactive test run
*/
   public void paint(Graphics g){
      if(!m_initializing && m_drawAON){
         if(m_displaySpringLayout){
            paintAONSpringLayout(g);
         }else{
            paintAON(g);
         }//else
      }//if
   }//paint

   /* This function draws the agents and the tasks altering the locations of
*/
   /*   the agents by treating the network connections as springs pulling
*/
   /*   agents closer to each other if they are connected by a network
*/
   /*   connection
*/
   public void paintAONSpringLayout(Graphics g){
      g.setColor(Color.white);
      g.fillRect(0,0,(int)this.getWidth(),(int)this.getHeight());
      paintRoads(g);
      paintTasks(g);
      for(int i = 0; i < getLength(m_agents); i++){
         AgentX agent = (AgentX)getItem(i,m_agents);

agent.paintAgentSpringLayout(m_columnOffset,m_rowOffset,m_ovalWidth,m_drawTaskI
Ds,m_drawAgentIDs,g);
      }//for

      if(m_drawStatusText){
         g.setColor(Color.black);

g.drawString("iter="+String.valueOf(m_iteration),m_columnOffset,m_rowOffset);

g.drawString("perf="+String.valueOf(m_performance),m_columnOffset,m_rowOffset+1
5);

g.drawString("eff="+String.valueOf(m_efficiency),m_columnOffset,m_rowOffset+30)
;
```

```
g.drawString("ave_deg="+String.valueOf(m_averageDegree),m_columnOffset,m_rowOff
set+45);
         g.drawString("wait="+String.valueOf(m_timeWaiting)+"
rewire="+String.valueOf(m_timeRewiring)
            +" propose="+String.valueOf(m_timeProposing)+"
join="+String.valueOf(m_timeJoining)
            +"
work="+String.valueOf(m_timeWorking),m_columnOffset,m_rowOffset+60);
      }//if
      setTitle("TestAON " + m_description + " " + m_runNumber);
   }//paintAONSpringLayout

   /* This function draws the agents and the tasks drawing the agents in
*/
   /*   their actual pixel positions
*/
   public void paintAON(Graphics g){
      g.setColor(Color.white);
      g.fillRect(0,0,(int)this.getWidth(),(int)this.getHeight());
      paintRoads(g);
      paintTasks(g);
      for(int i = 0; i < getLength(m_agents); i++){
         AgentX agent = (AgentX)getItem(i,m_agents);

agent.paintAgent(m_columnOffset,m_rowOffset,m_ovalWidth,m_drawTaskIDs,m_drawAge
ntIDs,g);
      }//for

      if(m_drawStatusText){
         g.setColor(Color.black);

g.drawString("iter="+String.valueOf(m_iteration),m_columnOffset,m_rowOffset);

g.drawString("perf="+String.valueOf(m_performance),m_columnOffset,m_rowOffset+1
5);

g.drawString("eff="+String.valueOf(m_efficiency),m_columnOffset,m_rowOffset+30)
;

g.drawString("ave_deg="+String.valueOf(m_averageDegree),m_columnOffset,m_rowOff
set+45);
         g.drawString("wait="+String.valueOf(m_timeWaiting)+"
rewire="+String.valueOf(m_timeRewiring)
            +" propose="+String.valueOf(m_timeProposing)+"
join="+String.valueOf(m_timeJoining)
            +"
work="+String.valueOf(m_timeWorking),m_columnOffset,m_rowOffset+60);
      }//if
      setTitle("TestAON " + m_description + " " + m_runNumber);
   }//paintAON

   /* This function draws the tasks at their pixel locations
*/
   private void paintTasks(Graphics g){
      for(int i = m_tasks.getFirstCurrentIndex(); i < m_tasks.getLength();
i++){
         AONTask task = (AONTask)m_tasks.getItem(i);
         if(!task.isExpired() && task.getState() != C_TASK_COMPLETE){
            int x = task.getX(), y = task.getY();
            int rectXOffset = m_columnOffset-((int)m_ovalWidth/2);
            int rectYOffset = m_rowOffset-((int)m_ovalWidth/2);
            g.setColor(Color.black);
```

```
            g.fillRect(x+rectXOffset,y+rectYOffset,m_ovalWidth,m_ovalWidth);
            int ovalXOffset = m_columnOffset-m_taskRange;
            int ovalYOffset = m_rowOffset-m_taskRange;
            g.setColor(Color.red);

g.drawOval(x+ovalXOffset,y+ovalYOffset,2*m_taskRange,2*m_taskRange);
         }//if
      }//for
      for(int i = m_tasks.getFirstCurrentIndex(); i < m_tasks.getLength();
i++){
         AONTask task = (AONTask)m_tasks.getItem(i);
         if(!task.isExpired()){
            int x = task.getX(), y = task.getY();
            int rectXOffset = m_columnOffset-((int)m_ovalWidth/2);
            int rectYOffset = m_rowOffset-((int)m_ovalWidth/2);
            g.setColor(Color.black);
            g.fillRect(x+rectXOffset,y+rectYOffset,m_ovalWidth,m_ovalWidth);
            int textXOffset = m_columnOffset+m_ovalWidth/2, textYOffset =
m_rowOffset+m_ovalWidth/2;

g.drawString(String.valueOf(task.getID()),x+textXOffset,y+textYOffset);
         }//if
      }//for
   }//paintTasks

   /* This function draws the roads in the background behind the agents and
*/
   /*   tasks
*/
   private void paintRoads(Graphics g){
      if(m_useRoads){
         m_roads.paint(g,m_columnOffset,m_rowOffset);
      }//if
   }//paintRoads

   /* This function draws the current status of the agents, their network
*/
   /*   connections and the tasks and saves the drawing to a JPEG image
*/
   private void performScreenCapture(String outputFilename, boolean
springLayout){
      Rectangle captureSize = new Rectangle(this.getBounds());
      BufferedImage bufferedImage = null;

      bufferedImage = new
BufferedImage((int)captureSize.getWidth(),(int)captureSize.getHeight(),Buffered
Image.TYPE_INT_RGB);
      if(springLayout){
         paintAONSpringLayout((Graphics)bufferedImage.getGraphics());
      }else{
         paintAON((Graphics)bufferedImage.getGraphics());
      }//else

      try{
         ImageIO.write(((RenderedImage)bufferedImage), "JPEG", new
File(outputFilename));
      }catch(IOException ex){
         System.out.println("Error writing screenshot: " + ex);
      }//catch
   }//performScreenCapture

   /* This function draws the current status of the agents, their network
*/
```

```
   /*   connections and the tasks and saves the drawing to a JPEG image
*/
   private void performScreenCapture(String outputFilename, int width, int
height){
      BufferedImage bufferedImage = bufferedImage = new
BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);
      paintAON((Graphics)bufferedImage.getGraphics());

      try{
         ImageIO.write(((RenderedImage)bufferedImage), "JPEG", new
File(outputFilename));
      }catch(IOException ex){
         System.out.println("Error writing screenshot: " + ex);
      }//catch
   }//performScreenCapture

   /* This function initializes the Agent-organized network for the test
*/
   public void initializeAON(){
      m_initializing = true;
      //Check to see if agents have been loaded from the xml input file
      if(getLength(m_agents) == 0){
         //Create agents
         System.out.println(m_agentTypes);
         createAgents();
         writeAgents();
         m_maxNodeDegree = getLength(m_agents);
      }else{
         m_agentCount = getLength(m_agents);
      }//else

      if(m_autoNetworkAgents){
         networkAgents();
      }//if

      if(m_useRoads){
         for(int i = 0; i < m_agentCount; i++){
            AgentX agent = (AgentX)getItem(i,m_agents);
            int shape_id = m_roads.getNearestShape(new
Point(agent.getX(),agent.getY()));
            if(shape_id <= 0){
               shape_id = m_roads.getRandomShape();
            }//if
            Point p = m_roads.getRandomLocation(shape_id);
            agent.setX((int)p.getX());
            agent.setY((int)p.getY());
         }//for
      }//if

      for(int i = 0; i < m_agentCount; i++){
         AgentX agent = (AgentX)getItem(i,m_agents);
         writeLog(agent.toString());
      }//for

      m_agentSkillHistogram = getAgentSkillHistogram();
      System.out.println(m_agentSkillHistogram);

      m_initializing = false;
      generateInitialTasks();

      printAgents();
   }//initializeAON
```

```
   /* This function adds network connections between the agents
*/
   private void networkAgents(){
      //Network agents
      for(int i = 0; i < m_agentCount; i++){
         AgentX agent = (AgentX)getItem(i,m_agents);
         networkAgent(i,agent);
      }//for

      //Add more connections so everyone has at least two
      for(int i = 0; i < m_agentCount; i++){
         AgentX agent = (AgentX)getItem(i,m_agents);
         int ctNeighbor = agent.getDegree();
         while(ctNeighbor < 2){
            int j = rand.nextInt(m_agentCount);
            if (i!=j)  agent.addNeighbor(j);
            ctNeighbor = agent.getDegree();
         }//while
      }//for
   }//networkAgents

   /* This function adds connections between the specified agent and other
*/
   /*   agents within m_maxInitialDistance of the agent's location
*/
   private void networkAgent(int index, AgentX agent){
      int x1 = agent.getX(), y1 = agent.getY();
      for(int j = 0; j < m_agentCount; j++){
         if(index != j){
            AgentX neighbor = (AgentX)getItem(j,m_agents);
            int x2 = neighbor.getX(), y2 = neighbor.getY();
            int distance = (int)Math.sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
            if(distance < m_maxInitialDistance){
               agent.addNeighbor(j);
            }//if
         }//if
      }//for
   }//networkAgent

   /* This function returns a random agent type
*/
   public int getAgentType(){
      float value = rand.nextFloat();
      for(int i = 0; i < getLength(m_agentTypeRatio); i++){
         if(value < (Float)getItem(i,m_agentTypeRatio)){
            return i;
         }//if
      }//for
      return getLength(m_agentTypeRatio) - 1;
   }//getAgentType

   /* This function sets the location for the specified agent for the test
*/
   private Point setAgentLocation(AgentX agent){
      int x, y;
      int sign = 1;
      if(rand.nextFloat() < 0.5f)sign = -1;
      if(m_clusterAgents){
         Point cluster =
(Point)getItem(rand.nextInt(getLength(m_agentCluster)),m_agentCluster);
         x = (int)cluster.getX() + sign*rand.nextInt(50);
         y = (int)cluster.getY() + sign*rand.nextInt(50);
      }else if(m_useRoads){
```

```
         int id = m_roads.getRandomShape();
         Point location = m_roads.getRandomLocation(id);
         x = (int)location.getX();
         y = (int)location.getY();
         agent.setShapeID(id);
      }else{
         x = rand.nextInt(m_columnCount-2*m_columnOffset);
         y = rand.nextInt(m_rowCount-2*m_rowOffset);
      }//else
      agent.setX(x);
      agent.setY(y);
      return new Point(x,y);
   }//setAgentLocation

   /* This function creates new agents for the test
*/
   public void createAgents(){
      for(int i = 0; i < m_agentCount; i++){
         int x = 0, y = 0;
         int agentTypeIndex = getAgentType();
         int count =
((Integer)getItem(agentTypeIndex,m_agentTypeCount)).intValue() + 1;
         setItem(agentTypeIndex,m_agentTypeCount,new Integer(count));
         String type = (String)getItem(agentTypeIndex,m_agentTypes);
         AgentX agent = addAgent(i,x,y,type);
         Point location = setAgentLocation(agent);
         initializeAgent(agent,agentTypeIndex);
      }//for
   }//createAgents

   /* This function initializes the attributes for the specified agent
*/
   private void initializeAgent(AgentX agent, int agentTypeIndex){
      Vector<Integer> agentSkills =
(Vector)getItem(agentTypeIndex,m_agentTypeSkills);
      int skill =
((Integer)getItem(rand.nextInt(getLength(agentSkills)),agentSkills)).intValue()
;
      if(skill >= 0){
         agent.addSkill(skill);
      }else{
         if(m_useCommandSkills){
            skill = rand.nextInt(m_numberOfSkills-1)+1;//exclude command skill
         }else{
            skill = rand.nextInt(m_numberOfSkills);
         }//else
         agent.addSkill(skill);
      }//else
      agent.setMaxActiveTime(m_maxActiveTime);
      agent.setMaxCommittedTime(m_maxCommittedTime);
      agent.setMaxNodeDegree(m_maxNodeDegree);
      agent.setCommunicationDepth(m_communicationDepth);
      agent.setNeighborhoodRadius(m_neighborhoodRadius);
      agent.setTaskRange(m_taskRange);
      agent.setAdaptNetwork(false);
      agent.setAdaptationRate(m_adaptationRate);
      agent.setMoveToTask(m_useTaskRange);
      agent.setSystemSkillsCount(m_numberOfSkills);
      agent.setSkillHistoryLength(m_skillHistoryLength);
      if(m_useGlobalRandomSeed){
         agent.setRandomSeed(m_randomSeed);
      }//if
      if(m_useRoads){
```

```
         agent.setRoads(m_roads);
      }//if
   }//initializeAgent

   /* This function initializes the attributes for the specified agent
*/
   private void initializeAgent(AgentX agent){
      int skill = rand.nextInt(m_numberOfSkills);
      agent.addSkill(skill);
      agent.setMaxActiveTime(m_maxActiveTime);
      agent.setMaxCommittedTime(m_maxCommittedTime);
      agent.setMaxNodeDegree(m_maxNodeDegree);
      agent.setCommunicationDepth(m_communicationDepth);
      agent.setNeighborhoodRadius(m_neighborhoodRadius);
      agent.setTaskRange(m_taskRange);
      agent.setAdaptNetwork(false);
      agent.setAdaptationRate(m_adaptationRate);
      agent.setMoveToTask(m_useTaskRange);
      agent.setSystemSkillsCount(m_numberOfSkills);
      agent.setSkillHistoryLength(m_skillHistoryLength);
      if(m_useGlobalRandomSeed){
         agent.setRandomSeed(m_randomSeed);
      }//if
      if(m_useRoads){
         agent.setRoads(m_roads);
      }//if
   }//initializeAgent

   /* This function creates a new agent of the specified agent type and adds
*/
   /*   the agent to the list of agents for this test
*/
   public AgentX addAgent(int id, int x, int y, String type){
      if(type.compareToIgnoreCase("Structural") == 0){
         StructuralAgent agent = new StructuralAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("StructuralImpatient") == 0){
         StructuralImpatientAgent agent = new
StructuralImpatientAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("StructuralStrat") == 0){
         StructuralStratAgent agent = new
StructuralStratAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("StructuralStratImpatient") == 0){
         StructuralStratImpatientAgent agent = new
StructuralStratImpatientAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("Egalitarian") == 0){
         EgalitarianAgent agent = new EgalitarianAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("EgalitarianImpatient") == 0){
         EgalitarianImpatientAgent agent = new
EgalitarianImpatientAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("EgalitarianStrat") == 0){
```

```
         EgalitarianStratAgent agent = new
EgalitarianStratAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("EgalitarianStratImpatient") == 0){
         EgalitarianStratImpatientAgent agent = new
EgalitarianStratImpatientAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("Performance") == 0){
         PerformanceAgent agent = new PerformanceAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("PerformanceImpatient") == 0){
         PerformanceImpatientAgent agent = new
PerformanceImpatientAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("PerformanceStrat") == 0){
         PerformanceStratAgent agent = new
PerformanceStratAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("PerformanceStratImpatient") == 0){
         PerformanceStratImpatientAgent agent = new
PerformanceStratImpatientAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("Diversity") == 0){
         DiversityAgent agent = new DiversityAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("DiversityImpatient") == 0){
         DiversityImpatientAgent agent = new
DiversityImpatientAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("DiversityStrat") == 0){
         DiversityStratAgent agent = new
DiversityStratAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("DiversityStratImpatient") == 0){
         DiversityStratImpatientAgent agent = new
DiversityStratImpatientAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("Inventory") == 0){
         InventoryAgent agent = new InventoryAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("InventoryImpatient") == 0){
         InventoryImpatientAgent agent = new
InventoryImpatientAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("InventoryStrat") == 0){
         InventoryStratAgent agent = new
InventoryStratAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("InventoryStratImpatient") == 0){
```

```
         InventoryStratImpatientAgent agent = new
InventoryStratImpatientAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("Uniform") == 0){
         UniformAgent agent = new UniformAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("UniformImpatient") == 0){
         UniformImpatientAgent agent = new
UniformImpatientAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("UniformStrat") == 0){
         UniformStratAgent agent = new UniformStratAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else if(type.compareToIgnoreCase("UniformStratImpatient") == 0){
         UniformStratImpatientAgent agent = new
UniformStratImpatientAgent(this,id,x,y,m_agents);
         m_agents.add(agent);
         return agent;
      }else{
         AgentX agent = new AgentX(this,id,x,y,"AgentX",m_agents);
         m_agents.add(agent);
         return agent;
      }//else
   }//addAgent

   /* This function creates a new agent of the specified type and inserts the
*/
   /*   agent into the list of agents for the tes
*/
   public AgentX insertAgent(int id, int x, int y, String type){
      if(type.compareToIgnoreCase("Structural") == 0){
         StructuralAgent agent = new StructuralAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("StructuralImpatient") == 0){
         StructuralImpatientAgent agent = new
StructuralImpatientAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("StructuralStrat") == 0){
         StructuralStratAgent agent = new
StructuralStratAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("StructuralStratImpatient") == 0){
         StructuralStratImpatientAgent agent = new
StructuralStratImpatientAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("Egalitarian") == 0){
         EgalitarianAgent agent = new EgalitarianAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("EgalitarianImpatient") == 0){
         EgalitarianImpatientAgent agent = new
EgalitarianImpatientAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("EgalitarianStrat") == 0){
```

```
         EgalitarianStratAgent agent = new
EgalitarianStratAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("EgalitarianStratImpatient") == 0){
         EgalitarianStratImpatientAgent agent = new
EgalitarianStratImpatientAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("Performance") == 0){
         PerformanceAgent agent = new PerformanceAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("PerformanceImpatient") == 0){
         PerformanceImpatientAgent agent = new
PerformanceImpatientAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("PerformanceStrat") == 0){
         PerformanceStratAgent agent = new
PerformanceStratAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("PerformanceStratImpatient") == 0){
         PerformanceStratImpatientAgent agent = new
PerformanceStratImpatientAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("Diversity") == 0){
         DiversityAgent agent = new DiversityAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("DiversityImpatient") == 0){
         DiversityImpatientAgent agent = new
DiversityImpatientAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("DiversityStrat") == 0){
         DiversityStratAgent agent = new
DiversityStratAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else if(type.compareToIgnoreCase("DiversityStratImpatient") == 0){
         DiversityStratImpatientAgent agent = new
DiversityStratImpatientAgent(this,id,x,y,m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }else{
         AgentX agent = new AgentX(this,id,x,y,"AgentX",m_agents);
         setItem(id,m_agents,agent);
         return agent;
      }//else
   }//insertAgent

   /* This function saves the list of agents for the test to an XML file
*/
   public void writeAgents(){
      XMLElement root = new XMLElement("AON",null);
      XMLElement xmlAgents = new XMLElement("Agents",root);
      for(int i = 0; i < getLength(m_agents); i++){
         AgentX agent = (AgentX)getItem(i,m_agents);
         XMLElement xmlAgent = new XMLElement("Agent"+agent.getID(),xmlAgents);
         xmlAgent.addAttribute("ID",String.valueOf(agent.getID()));
```

```
         xmlAgent.addAttribute("X",String.valueOf(agent.getX()));
         xmlAgent.addAttribute("Y",String.valueOf(agent.getY()));
         xmlAgent.addAttribute("Type",agent.getType());

         int shapeID = agent.getShapeID();
         if(shapeID > 0){
            xmlAgent.addAttribute("ShapeID",String.valueOf(shapeID));
         }//if

         XMLElement xmlSkills = new XMLElement("Skills",xmlAgent);
         XMLElement xmlSkill = new XMLElement("Skill0",xmlSkills);
         int skillID = agent.m_skill;
         xmlSkill.addAttribute("SkillID",String.valueOf(skillID));
         xmlSkills.addElement("Skill0",xmlSkill);

         xmlAgent.addElement("Skills",xmlSkills);
         xmlAgents.addElement("Agent"+agent.getID(),xmlAgent);
      }//for
      root.addElement("Agents",xmlAgents);
      AONXMLReader xmlFile = new AONXMLReader();
      xmlFile.setRoot(root);
      if(xmlFile.writeXML("AON_Agents.xml")){
         System.out.println("Wrote AON_Agents.xml");
      }else{
         System.out.println("Could not write AON_Agents.xml");
      }//else
   }//writeAgents

   /* This function updates the list of tasks that are available for agents
*/
   /*   to work on
*/
   public void advertiseTask(){
      for(int i = m_tasks.getFirstCurrentIndex(); i < m_tasks.getLength();
i++){
         AONTask task = (AONTask)m_tasks.getItem(i);
         if(!task.isExpired()){
            task.updateAdvertised();
            if(task.isExpired()){
               int state = task.getState();
               switch(state){
                  case(C_PENDING):
                      m_pendingExpiredTasks++;
                      break;
                  case(C_SKILLS_FAIL):
                     m_skillFailExpiredTasks++;
                     m_unfilledSkillsCount =
task.countUnfilledSkills(m_unfilledSkillsCount);
                     break;
                  case(C_DISTANCE_FAIL):
                     m_distanceFailExpiredTasks++;
                     break;
                  case(C_TASK_COMPLETE):
                     m_completeExpiredTasks++;
                     break;
                  default:
               }//switch
            }//if
         }//if
      }//for

      m_tasks.updateFirstCurrentIndex();
```

```
      if(m_iteration % m_taskAnnounceTime == 0){
         for(int i = 0; i < m_numberOfTasksPerBatch; i++){
            generateTask(m_maxAdvertisedTime);
         }//for
         computeOptimal();
      }//if
      m_agentSkillPriorityList.updateAllListAgents();
   }//advertiseTask

   /* This function generates a task with the specified maximum advertised
*/
   /*   time that the agents can work on
*/
   public void generateTask(int maxAdvertisedTime){
      if(m_clusterTasks){
         for(int i = 0; i < getLength(m_taskCluster); i++){
            generateClusteredTask(maxAdvertisedTime,i);
         }//for
      }else{
         generateRandomDistributedTask(maxAdvertisedTime);
      }//else
   }//generateTask

   /* This function generates a task that is clustered near one of the task
*/
   /*   cluster locations
*/
   public void generateClusteredTask(int maxAdvertisedTime, int clusterIndex){
      TaskCluster cluster = (TaskCluster)getItem(clusterIndex,m_taskCluster);
      if(cluster.getTasksLeftCount() <= 0 ||
!cluster.timeStepValid(m_iteration)){
         return;
      }//if
      if(m_clusterSpread){
         TaskCluster newCluster = cluster.spread();
         if(newCluster != null){
            m_taskCluster.add(newCluster);
            m_clusterCount = getLength(m_taskCluster);
         }//if
      }//if
      Point clusterPoint = cluster.getPoint(m_iteration);
      Vector<Float> skillWeights = cluster.getSkillWeights();
      int sign = 1;
      if(rand.nextFloat() < 0.5f)sign = -1;
      int x = (int)clusterPoint.getX() + sign*rand.nextInt(50);
      int y = (int)clusterPoint.getY() + sign*rand.nextInt(50);
      AONTask task = new
AONTask(this,m_taskID,x,y,maxAdvertisedTime,m_maxActiveTime,m_numberOfSkills);/
/,m_numberOfSkillsPerTask,m_numberOfSkills);
      task.setClusterIndex(clusterIndex);
      task.setSkillWeights(skillWeights);
      initializeTask(task);
      cluster.addTaskCount();
      m_taskID++;
      m_tasks.add(task);
      m_taskCount++;
      m_numberOfTasksIntroduced++;
   }//generateClusteredTask

   /* This function generates a task that is located at a random location in
*/
   /*   the test area
*/
```

```
   public void generateRandomDistributedTask(int maxAdvertisedTime){
      int x = 0, y = 0;
      if(m_useRoads){
         Point location = m_roads.getRandomTaskLocation(m_taskRange);
         x = (int)location.getX();
         y = (int)location.getY();
      }else{
         x = rand.nextInt(m_columnCount-2*m_columnOffset);
         y = rand.nextInt(m_rowCount-2*m_rowOffset);
      }//else
      AONTask task = new
AONTask(this,m_taskID,x,y,maxAdvertisedTime,m_maxActiveTime,m_numberOfSkills);
      initializeTask(task);
      m_taskID++;
      m_tasks.add(task);
      m_taskCount++;
      m_numberOfTasksIntroduced++;
   }//generateRandomDistributedTask

   /* This function initializes a new task with the appropriate settings
*/
   public void initializeTask(AONTask task){
      task.setUseCommandSkills(m_useCommandSkills);
      task.setUseTaskRange(m_useTaskRange);
      task.addSkills(m_numberOfSkillsPerTask,m_numberOfSkills);
      m_skillDemandCount = task.countSkills(m_skillDemandCount);
      Hashtable availableSkillHistogram = getAvailableAgentSkillHistogram();

if(task.taskImpossible(availableSkillHistogram)){//m_agentSkillHistogram))
         m_impossibleTaskCount++;
      }//if
      if(m_useGlobalRandomSeed){
         task.setRandomSeed(m_randomSeed);
      }//if
   }//initializeTask

   /* This function writes a report showing the number of agents that have
*/
   /*   each possible number of network connections to the log file
*/
   public void logNodeDegreeInformation(){
      final int DEGMAX=40;
      int totalDeg=0;
      int [] tallyDeg = new int [DEGMAX];
      for (int i=0; i < getLength(m_agents);i++){
         int deg = ((AgentX)getItem(i,m_agents)).getDegree();
         totalDeg+=deg;
         if(deg >= DEGMAX) deg = DEGMAX-1;
         tallyDeg[deg]++;
      }//for
      String results = "Degree Tallies";
      for (int i = 0; i < DEGMAX; i++){
         results+= i+":" +tallyDeg[i] + " ";
      }//for
      writeLog(results);
      writeLog("totalDegree" + totalDeg);
   }//logNodeDegreeInformation

   /* This function updates the agents and their network connections for the
*/
   /*   current time step
*/
   public void evolveAON(){
```

```
      writeLog("**ITERATION ** "+ m_iteration);
      if(m_iteration <= m_maximumIterations){
         createAgentProcessingList();
         int agentDegreeSum = 0, agentDensitySum = 0;
         m_minDegree = 9999;
         m_maxDegree = 0;
         m_joinedImpossibleTaskCount = 0;
         m_proposedImpossibleTaskCount = 0;
         m_repeatTaskCount = 0;
         m_neighborhoodSkillShortageSum = 0;
         m_minNeighborhoodSkillShortage = 9999;
         m_maxNeighborhoodSkillShortage = 0;
         m_aliveAgentCount = 0;
         int committedCount = 0, taskDistanceSum = 0;
         for(int i = 0; i < getLength(m_agents); i++){
            int id =
((Integer)m_agentProcessingList.remove(rand.nextInt(getLength(m_agentProcessing
List)))).intValue();
            checkForNodeFailure(id);
            checkForNodeRegeneration(id);
            AgentX agent = (AgentX)getItem(id,m_agents);
            if(m_iteration >= m_minAdaptTimeStep && m_iteration <
m_maxAdaptTimeStep){
               agent.setAdaptNetwork(true);
            }else{
               agent.setAdaptNetwork(false);
            }//else
            agent.update(m_tasks);
            if(agent.getState() == C_COMMITTED){
               taskDistanceSum = taskDistanceSum + agent.getTaskDistance();
               committedCount++;
            }//if
            if(agent.getState() >= 0){
               m_aliveAgentCount++;
            }//if
            m_changedEdges = m_changedEdges + agent.getChangedEdgesCount();
            countAgentActivity(agent);
            countTaskSuccess(agent);
            countTaskCommitments(agent);
            countSkillShortage(agent);
            countDistanceTravelled(agent);

            int degree = agent.getDegree();
            int density = agent.getDensity(3);
            agentDegreeSum = agentDegreeSum + degree;
            agentDensitySum = agentDensitySum + density;
            if(degree < m_minDegree){
               m_minDegree = degree;
            }//if
            if(degree > m_maxDegree){
               m_maxDegree = degree;
            }//if
         }//for
         computeAONStats(agentDegreeSum,agentDensitySum,taskDistanceSum);
         if(m_iteration % m_writeDataInterval == 0 || m_iteration ==
m_minAdaptTimeStep || m_iteration == m_maximumIterations){
            writeData();
         }//if
         m_iteration++;
      }else{
         stopAON();
      }//else
   }//evolveAON
```

```
   /* This function checks to see if it is time to increase the number of
*/
   /*   tasks to be created per time step
*/
   private void checkTaskSpike(){
      int taskSpikeCount = getLength(m_taskSpikes);
      for(int i = 0; i < taskSpikeCount; i++){
         TaskSpike taskSpike = (TaskSpike)getItem(i,m_taskSpikes);
         int timeStep = taskSpike.getTimeStep(), duration =
taskSpike.getDuration();
         if(timeStep == m_iteration){
            System.out.println("Start Task Spike");
            taskSpike.saveTasksPerBatch(m_numberOfTasksPerBatch);
            taskSpike.saveTaskAnnounceTime(m_taskAnnounceTime);
            processTaskAnnounceTime(taskSpike.getTaskAnnounceTime());
         }else if(timeStep <= m_iteration){
            if(duration == 0){
               System.out.println("End Task Spike");
               m_numberOfTasksPerBatch = taskSpike.retrieveTasksPerBatch();
               m_taskAnnounceTime = taskSpike.retrieveTaskAnnounceTime();
            }//if
            if(duration >= 0){
               taskSpike.decrementDuration();
            }//if
         }//else if
      }//for
   }//checkTaskSpike

   /* This function creates a list of agent ids to be processed, allowing the
*/
   /*   agents to be removed from this list at random until no agents are
*/
   /*   left to be updated in EvolveAON.  This list is created at each
*/
   /*   timestep
*/
   private void createAgentProcessingList(){
      if(getLength(m_agentProcessingList) == 0){
         for(int i = 0; i < getLength(m_agents); i++){
            m_agentProcessingList.add(new Integer(i));
         }//for
      }//if
   }//createAgentProcessingList

   /* This function checks whether a node needs to be selected for failure
*/
   private void checkForNodeFailure(int id){
      for(int i = 0; i < getLength(m_nodeFailure); i++){
         NodeFailure failure = (NodeFailure)getItem(i,m_nodeFailure);
         int timeStep = failure.getTimeStep(), agentID = failure.getAgentID();
         if(agentID == id && timeStep == m_iteration){
            writeLog("Node failed: agentID=" + agentID + " timeStep=" +
timeStep);
            m_failedNodes.add(agentID);
            AgentX failedAgent = (AgentX)getItem(id,m_agents);
            setItem(id,m_agents,new FailedAgentX(failedAgent));
         }//if
      }//for
   }//checkForNodeFailure

   /* This function checks whether a node is due to be regenerated after it
*/
```

```
   /*   has previously failed
*/
   private void checkForNodeRegeneration(int id){
      for(int i = 0; i < getLength(m_nodeFailure); i++){
         NodeFailure failure = (NodeFailure)getItem(i,m_nodeFailure);
         int timeStep = failure.getTimeStep(), agentID = failure.getAgentID(),
regenerationTime = failure.getRegenerationTime();
         if(agentID == id && timeStep <= m_iteration){
            if(regenerationTime == 0){
               writeLog("Node regenerated: agentID=" + agentID + " timeStep=" +
timeStep);
               FailedAgentX failedAgent = (FailedAgentX)getItem(id,m_agents);
               setAgentLocation(failedAgent.getDeadAgent());
               networkAgent(id,failedAgent.getDeadAgent());
               setItem(id,m_agents,failedAgent.getDeadAgent());
               m_failedNodes.removeElement(agentID);
            }//if
            if(regenerationTime >= 0){
               failure.decrementRegenerationTime();
            }//if
         }//if
      }//for
   }//checkForNodeRegeneration

   /* This function updates the global counts of how long all agents have
*/
   /*   spent in each work catagory
*/
   private void countAgentActivity(AgentX agent){
      int workCatagory = agent.getWorkCatagory();
      switch(workCatagory){
         case C_WAITED:
            m_timeWaiting++;
            break;
         case C_REWIRED:
            m_timeRewiring++;
            break;
         case C_PROPOSED:
            m_timeProposing++;
            break;
         case C_JOINED:
            m_timeJoining++;
            break;
         case C_WORKED:
            m_timeWorking++;
            break;
         default:
            break;
      }//switch
   }//countAgentActivity

   /* This function checks whether the specified agent's team has formed
*/
   /*   successfully
*/
   private void countTaskSuccess(AgentX agent){
      if(agent.getState() == C_ACTIVE){
         String teamString = agent.getTeamString();
         if(!contains(teamString,m_teamsFormed)){
            m_teamsFormed.add(teamString);
            m_numberOfTeamsFormed++;
            AONTask task = agent.getTask();
            if(task != null){
```

```
               String stateString = "";
               Vector <Integer>agentsCommitted = task.getAgentsCommitted();
               for(int i = 0; i < getLength(agentsCommitted); i++){
                  int index = ((Integer)getItem(i,agentsCommitted)).intValue();
                  AgentX committedAgent = (AgentX)getItem(index,m_agents);
                  committedAgent.setState(C_ACTIVE);
               }//for
               task.setSuccessful();
            }//ifn
            int clusterIndex = task.getClusterIndex();
            if(clusterIndex >= 0){
               TaskCluster cluster =
(TaskCluster)getItem(clusterIndex,m_taskCluster);
               cluster.addSuccessCount();
            }//if
            countAgentTypeUtility(agent);
         }//if
         m_numberOfAgentsActive++;
      }//if
   }//countTaskSuccess

   /* This function updates the counts for how many impossible tasks have
*/
   /*   been joined, proposed, and how many tasks have been committed to
*/
   /*   repeatedly
*/
   private void countTaskCommitments(AgentX agent){
      m_joinedImpossibleTaskCount += agent.getJoinedImpossibleCount();
      m_proposedImpossibleTaskCount += agent.getProposedImpossibleCount();
      m_repeatTaskCount += agent.getRepeatTaskCount();
   }//countTaskCommitments

   /* This function updates the count of how many skills each neighborhood
*/
   /*   is lacking
*/
   private void countSkillShortage(AgentX agent){
      int skillShortage = agent.getNeighborhoodSkillShortage();
      if(skillShortage < m_minNeighborhoodSkillShortage){
         m_minNeighborhoodSkillShortage = skillShortage;
      }//if
      if(skillShortage > m_maxNeighborhoodSkillShortage){
         m_maxNeighborhoodSkillShortage = skillShortage;
      }//if
      m_neighborhoodSkillShortageSum += skillShortage;
   }//countSkillShortage

   /* This function updates the count for the distance agents have travelled
*/
   private void countDistanceTravelled(AgentX agent){
      m_distanceTravelled = m_distanceTravelled + agent.getDistanceTravelled();
   }//countDistanceTravelled

   /* This function updates the utility achieved by the agents for each agent
*/
   /*   type
*/
   private void countAgentTypeUtility(AgentX agent){
      if(agent.getState() == C_ACTIVE){
         String teamString = agent.getTeamString();
         if(!contains(teamString,m_teamsUtilityCounted)){
            m_teamsUtilityCounted.add(teamString);
```

```
            int taskID = agent.getTaskID();
            AONTask task = (AONTask)m_tasks.getItem(taskID);
            Vector agentsCommitted = task.getAgentsCommitted();
            for(int i = 0; i < getLength(agentsCommitted); i++){
               int index = ((Integer)getItem(i,agentsCommitted)).intValue();
               AgentX taskAgent = (AgentX)getItem(index,m_agents);
               int skillID = taskAgent.getSkills();
               float agentPayoff = task.getPayoff(skillID);
               String agentType = agent.getType();
               int agentTypeIndex = m_agentTypes.indexOf(agentType);
               float utility = 0.0f;
               try{
                  utility = (Float)getItem(agentTypeIndex,m_agentTypeUtility) +
agentPayoff;
               }catch(Exception ex){
                  System.out.println("agentTypeIndex=" + agentTypeIndex + "
agentType=" + agentType);
                  ex.printStackTrace();
               }//catch
               setItem(agentTypeIndex,m_agentTypeUtility,utility);
               m_totalAONUtility = m_totalAONUtility + agentPayoff;
            }//for
         }//if
      }//if
   }//countAgentTypeUtility

   /* This function computes some statistics for the current timestep of the
*/
   /*   test
*/
   private void computeAONStats(int agentDegreeSum, int agentDensitySum, float
taskDistanceSum){
      int meanDegree = (int)agentDegreeSum/getLength(m_agents);
      m_performance = (float)m_numberOfTeamsFormed/m_numberOfTasksIntroduced;
      m_numberOfAgentsUpdated = m_numberOfAgentsUpdated + m_agentCount;
      m_efficiency = (float)m_numberOfAgentsActive/m_numberOfAgentsUpdated;
      m_averageDegree = agentDegreeSum / getLength(m_agents);
      m_averageDensity = agentDensitySum / getLength(m_agents);
      int agentDegreeVarianceSum = 0;
      for(int i = 0; i < getLength(m_agents); i++){
         AgentX agent = (AgentX)getItem(i,m_agents);
         agentDegreeVarianceSum = agentDegreeVarianceSum + (agent.getDegree() -
meanDegree)*(agent.getDegree() - meanDegree);
         if(m_writeAgentData && m_iteration % m_writeDataInterval == 0){
            agent.writeAgentData(m_iteration,m_performance);
         }//if
      }//for
      m_degreeVariance = (float)agentDegreeVarianceSum / getLength(m_agents);
      m_averageTaskDistance = (float)taskDistanceSum / getLength(m_agents);
      m_averageNeighborhoodSkillShortage =
(float)m_neighborhoodSkillShortageSum / getLength(m_agents);
   }//computeAONStats

   /* This function updates the information displayed in the MessageFrame
*/
   /*   when the test is being run interactively
*/
   public void updateMessageFrame(){
      m_messageFrame.clearText();
      if(m_selectedAgent != null){
         m_messageFrame.setVisible(true);
         m_messageFrame.addText(m_selectedAgent.toString());
      }//if
```

```
      if(m_selectedTask != null){
         m_messageFrame.setVisible(true);
         m_messageFrame.addText(m_selectedTask.toString());
      }//if
   }//updateMessageFrame

   /* This function stops the test and writes the output files for the test
*/
   private void stopAON(){
      logNodeDegreeInformation();
      System.out.println("Stop...");
      m_updateTimer.stop();
      m_paintTimer.stop();
      repaint();
      writeAONRunFiles();
      if(m_exitWhenDone){
         m_running = false;
      }//if
   }//stopAON

   /* This function writes the statistics for the test to a CSV file
*/
   private void writeAONRunFiles(){
      if(checkDirectory()){
         m_outputFilename = m_outputDirectory + "/" + m_outputFilename + "_" +
m_runNumber + m_description + ".csv";
      }else{
         m_outputFilename = m_outputFilename + "_" + m_runNumber +
m_description + ".csv";
      }//else
      m_systemDataFile.save(m_outputFilename);
      if(m_writeAgentData){
         for(int i = 0; i < getLength(m_agents); i++){
            AgentX agent = (AgentX)getItem(i,m_agents);
            agent.writeAgentData(m_iteration,m_performance);
            String outputFilename = "";
            if(checkDirectory()){
               outputFilename = m_outputDirectory + "/AONAgent" + agent.getID()
+ "_" + m_runNumber + m_description + ".csv";
            }else{
               outputFilename = "/AONAgent" + agent.getID() + "_" + m_runNumber
+ m_description + ".csv";
            }//else
            agent.saveAgentData(outputFilename);
         }//for
      }//if
      writeSkillCountData();
      writeSkillFailureCountData();
      writeSkillDemandCountData();
   }//writeAONRunFiles

   /* This function writes the skill count data to a CSV file
*/
   private void writeSkillCountData(){
      CSVFile skillCountData = new CSVFile();
      String outputFilename = "";
      if(checkDirectory()){
         outputFilename = m_outputDirectory + "/ReportAONSkillCount_" +
m_description + ".csv";
      }else{
         outputFilename = "/ReportAONSkillCount_" + m_description + ".csv";
      }//else
```

```
      String header[] = {" ","skill_count"};
      skillCountData.addRow(header);

      System.out.println(m_agentSkillHistogram);
      for(int i = 0; i < m_numberOfSkills; i++){
         int agentCount =
(Integer)m_agentSkillHistogram.get(String.valueOf(i));
         Vector values = new Vector();
         values.add("Skill " + String.valueOf(i));
         values.add(String.valueOf(agentCount));
         skillCountData.addRow(values);
      }//for

      skillCountData.save(outputFilename);
   }//writeSkillCountData

   /* This function writes the skill failure data to a CSV file
*/
   private void writeSkillFailureCountData(){
      CSVFile skillCountData = new CSVFile();
      String outputFilename = "";
      if(checkDirectory()){
         outputFilename = m_outputDirectory + "/ReportAONSkillFailureCount_" +
m_description + ".csv";
      }else{
         outputFilename = "/ReportAONSkillFailureCount_" + m_description +
".csv";
      }//else

      String header[] = {" ","Failures_per_agent"};
      skillCountData.addRow(header);

      for(int i = 0; i < m_numberOfSkills; i++){
         int agentCount =
(Integer)m_agentSkillHistogram.get(String.valueOf(i));
         int unfilledCount = (Integer)getItem(i,m_unfilledSkillsCount);
         Vector values = new Vector();
         values.add("Skill " + String.valueOf(i));
         values.add(String.valueOf(unfilledCount));
         skillCountData.addRow(values);
      }//for

      skillCountData.save(outputFilename);
   }//writeSkillFailureCountData

   /* This function writes the skill demand data to a CSV file
*/
   private void writeSkillDemandCountData(){
      CSVFile skillDemandCountData = new CSVFile();
      String outputFilename = "";
      if(checkDirectory()){
         outputFilename = m_outputDirectory + "/ReportAONSkillDemandCount_" +
m_description + ".csv";
      }else{
         outputFilename = "/ReportAONSkillDemandCount_" + m_description +
".csv";
      }//else

      String header[] = {" ","skill_demand"};
      skillDemandCountData.addRow(header);

      System.out.println(m_agentSkillHistogram);
      for(int i = 0; i < m_numberOfSkills; i++){
```

```
         int agentCount = (Integer)getItem(i,m_skillDemandCount);
         Vector values = new Vector();
         values.add("Skill " + String.valueOf(i));
         values.add(String.valueOf(agentCount));
         skillDemandCountData.addRow(values);
      }//for

      skillDemandCountData.save(outputFilename);
   }//writeSkillDemandCountData

   /* This function writes a summary of the current agent state to the log
*/
   /*   file
*/
   private void agentSummary(){
      if(debug){
         int ctUn = 0, ctCom = 0, ctAct = 0;
         int ctPerf = 0, ctStruc = 0, ctStrucStrat = 0, ctStrucStratImpatient =
0, ctDiversityImpatient = 0;
         for(int i = 0; i < getLength(m_agents); i++){
            AgentX agent = (AgentX)getItem(i,m_agents);
            switch(agent.getState()){
               case(C_UNCOMMITTED):
                  ctUn++; break;
               case(C_COMMITTED):
                  ctCom++;break;
               case(C_ACTIVE):
                  ctAct++;
               default:
            }//switch
            String type = agent.getType();
            if(type.compareTo("Performance") == 0){
               ctPerf++;
            }else if(type.compareTo("Structural") == 0){
               ctStruc++;
            }else if(type.compareTo("StructuralStrat") == 0){
               ctStrucStrat++;
            }else if(type.compareTo("StructuralStratImpatient") == 0){
               ctStrucStratImpatient++;
            }else if(type.compareTo("DiversityImpatient") == 0){
               ctDiversityImpatient++;
            }//else if
         }//for

         writeLog(" Agent Status uncommitted " + ctUn + " committed " + ctCom +
" Active " + ctAct
                  + " Performance=" + ctPerf + " Structural=" + ctStruc + "
StrucStrat=" + ctStrucStrat
                  + " StrucStratImpatient=" + ctStrucStratImpatient + "
DiversityImpatient=" + ctDiversityImpatient);
      }//if
   }//agentSummary

   /* This function updates the tasks and the agents for the current time
*/
   /*   step, and is called either by the timer in an interactive test run,
*/
   /*   or from inside a continuous loop in a non-interactive test run
*/
   private void runTimeStep(){
      agentSummary();
      checkTaskSpike();
      advertiseTask();
```

```
      evolveAON();
      if(m_saveScreenShot && (m_iteration % m_screenShotInterval == 0 ||
m_iteration == 1)){
         String outputFilename = "";
         outputFilename = getScreenshotFilename(false);
         performScreenCapture(outputFilename,false);

         outputFilename = getScreenshotFilename(true);
         performScreenCapture(outputFilename,true);
      }//if
      for(int i = 0; i < getLength(m_agents); i++){
         AgentX agent = (AgentX)getItem(i,m_agents);
         agent.adjustLocation(m_columnCount-2*m_columnOffset, m_rowCount-
2*m_rowOffset);
      }//for
   }//runTimeStep

   /* This function returns a filename to be used to save a screenshot of the
*/
   /*   test that describes the current state of the test being pictured
*/
   private String getScreenshotFilename(boolean springLayout){
      String outputFilename = "AON";
      String springDescription = "";
      if(springLayout){
         springDescription = "spring";
      }//if
      if(checkDirectory()){
         outputFilename = m_outputDirectory + "/" + outputFilename + "_" +
m_runNumber + m_description + "_" + m_iteration + springDescription + ".jpg";
      }else{
         outputFilename = outputFilename + "_" + m_runNumber + m_description +
"_" + m_iteration + ".jpg";
      }//else
      return outputFilename;
   }//getScreenshotFilename

   /* This function returns a list of parameters being used in the test run
*/
   public String getParametersString(){
      return "N (agentCount)=" + m_agentCount + " alpha (activeTime)=" +
m_maxActiveTime + " gamma (advertisedTime)=" + m_maxAdvertisedTime
        + " sigma (Skills)=" + m_numberOfSkills + " |T|=" +
m_numberOfSkillsPerTask + " mu (announceTime)=" + m_taskAnnounceTime;
   }//getParametersString

   /* This function adds the data for the current time step to a row in the
*/
   /*   test's CSV file
*/
   public void writeData(){
      if(m_systemDataFile.getLineCount() == 0){
         String infoValues[] = {"AON Results " + getParametersString(), (new
Date()).toString()};
         m_systemDataFile.addRow(infoValues);

         Vector headerValues = new Vector();
         headerValues.add("iteration");
         headerValues.add("optimal_performance");
         headerValues.add("performance");
         headerValues.add("efficiency");
         headerValues.add("changed_edges");
         headerValues.add("average_degree");
```

```
      headerValues.add("average_density");
      headerValues.add("min_degree");
      headerValues.add("max_degree");
      headerValues.add("degree_variance");
      headerValues.add("time_waiting");
      headerValues.add("time_rewiring");
      headerValues.add("time_proposing");
      headerValues.add("time_joining");
      headerValues.add("time_working");
      headerValues.add("total_utility");
      headerValues.add("changed_agents");
      headerValues.add("average_task_distance");
      headerValues.add("pending_failures");
      headerValues.add("skill_failures");
      headerValues.add("distance_failures");
      headerValues.add("tasks_completed");
      headerValues.add("task_clusters");
      headerValues.add("task_count");
      headerValues.add("distance_travelled");
      headerValues.add("joined_impossible");
      headerValues.add("proposed_impossible");
      headerValues.add("repeat_task");
      headerValues.add("min_skill_distance");
      headerValues.add("max_skill_distance");
      headerValues.add("average_skill_distance");
      headerValues.add("alive_agents");
      for(int i = 0; i < getLength(m_agentTypes); i++){
         String agentType = (String)getItem(i,m_agentTypes);
         headerValues.add(agentType + "_count");
      }//for

      m_systemDataFile.addRow(headerValues);
   }//if
   Vector values = new Vector();
   values.add(String.valueOf(m_iteration));
   values.add(String.valueOf(m_optimalPerformance));
   values.add(String.valueOf(m_performance));
   values.add(String.valueOf(m_efficiency));
   values.add(String.valueOf(m_changedEdges));
   values.add(String.valueOf(m_averageDegree));
   values.add(String.valueOf(m_averageDensity));
   values.add(String.valueOf(m_minDegree));
   values.add(String.valueOf(m_maxDegree));
   values.add(String.valueOf(m_degreeVariance));
   values.add(String.valueOf(m_timeWaiting));
   values.add(String.valueOf(m_timeRewiring));
   values.add(String.valueOf(m_timeProposing));
   values.add(String.valueOf(m_timeJoining));
   values.add(String.valueOf(m_timeWorking));
   values.add(String.valueOf(m_totalAONUtility));
   values.add(String.valueOf(m_agentTypesChanged));
   values.add(String.valueOf(m_averageTaskDistance));
   values.add(String.valueOf(m_pendingExpiredTasks));
   values.add(String.valueOf(m_skillFailExpiredTasks));
   values.add(String.valueOf(m_distanceFailExpiredTasks));
   values.add(String.valueOf(m_completeExpiredTasks));
   values.add(String.valueOf(m_clusterCount));
   values.add(String.valueOf(m_taskCount));
   values.add(String.valueOf(m_distanceTravelled));
   values.add(String.valueOf(m_joinedImpossibleTaskCount));
   values.add(String.valueOf(m_proposedImpossibleTaskCount));
   values.add(String.valueOf(m_repeatTaskCount));
   values.add(String.valueOf(m_minNeighborhoodSkillShortage));
```

```
      values.add(String.valueOf(m_maxNeighborhoodSkillShortage));
      values.add(String.valueOf(m_averageNeighborhoodSkillShortage));
      values.add(String.valueOf(m_aliveAgentCount));
      for(int i = 0; i < getLength(m_agentTypes); i++){
         String agentType = (String)getItem(i,m_agentTypes);
         int count = ((Integer)getItem(i,m_agentTypeCount)).intValue();
         values.add(String.valueOf(count));
      }//for

      m_systemDataFile.addRow(values);
   }//writeData

   /* This function writes the specified string to the log file
*/
   public void writeLog(String line){
      if(!debug)return;
      try{
         BufferedWriter fileOut;
                           /*
*/
         boolean append = false;
         if(firstLog){
            firstLog=false;
            if(checkDirectory()){
               m_logFilename = m_outputDirectory + "/" + m_logFilename + "_" +
m_runNumber + m_description + ".txt";
            }else{
               m_logFilename = m_logFilename + "_" + m_runNumber +
m_description + ".txt";
            }//else
            fileOut = new BufferedWriter(new FileWriter(m_logFilename,append));
            fileOut.write("AON Log File " + (new Date()).toString());
            fileOut.newLine();
         }else{
            append = true;
            fileOut = new BufferedWriter(new FileWriter(m_logFilename,append));
         }//else

         fileOut.write(line);
         fileOut.newLine();
         fileOut.close();
      }catch(IOException ex){
         System.out.println("IOException while writing log: " +
ex.getMessage());
      }//catch
   }//writeLog

   /* This function checks to see if the output directory exists, attempting
*/
   /*   to create it if it does not exist
*/
   public boolean checkDirectory(){
      File outDirectory = new File(m_outputDirectory);
      if(!outDirectory.isDirectory()){
        return outDirectory.mkdir();
      }//if
      return outDirectory.exists();
   }//checkDirectory

   /* This function checks to see if the specified directory exists,
*/
   /*   attempting to create it if it does not exist
*/
```

```
   public boolean checkDirectory(String outputDirectory){
      File outDirectory = new File(outputDirectory);
      if(!outDirectory.isDirectory()){
         return outDirectory.mkdir();
      }//if
      return outDirectory.exists();
   }//checkDirectory

   /* This function returns the agent id for the agent with the most network
*/
   /*   connections
*/
   public int getHighestDegreeAgent(){
      int highDegreeAgentID = 0, maxDegree = 0;
      for(int i = 0; i < getLength(m_agents); i++){
         AgentX agent = (AgentX)getItem(i,m_agents);
         int degree = agent.getDegree();
         if(degree > maxDegree){
            highDegreeAgentID = i;
            maxDegree = degree;
         }//if
      }//for
      return highDegreeAgentID;
   }//getHighestDegreeAgent

   /* This function returns a set of key-value pairs respresenting the number
*/
   /*   of agents possessing each skill where the skill id is the key and the
*/
   /*   number of agents possessing the skill is the value
*/
   public Hashtable getAgentSkillHistogram(){
      Hashtable skillHistogram = new Hashtable();
      for(int i = 0; i < m_numberOfSkills; i++){
         String skillKey = String.valueOf(i);
         skillHistogram.put(skillKey,0);
      }//for
      for(int i = 0; i < getLength(m_agents); i++){
         AgentX agent = (AgentX)getItem(i,m_agents);
         int skill = agent.getSkills();
         String skillKey = String.valueOf(skill);
         int count = 1;
         if(skillHistogram.containsKey(skillKey)){
            int countSkill = (Integer)skillHistogram.get(skillKey);
            count += countSkill;
         }//if
         skillHistogram.put(skillKey, count);
      }//for

      return skillHistogram;
   }//getAgentSkillHistogram

   /* This function returns a set of key-value pairs representing the number
*/
   /*   of uncommitted agents possessing each skill where the skill id is the
*/
   /*   key and the number of uncommitted agents possessing the skill is the
*/
   /*   value
*/
   public Hashtable getAvailableAgentSkillHistogram(){
      Hashtable skillHistogram = new Hashtable();
      for(int i = 0; i < m_numberOfSkills; i++){
```

```
         String skillKey = String.valueOf(i);
         skillHistogram.put(skillKey,0);
      }//for
      for(int i = 0; i < getLength(m_agents); i++){
         AgentX agent = (AgentX)getItem(i,m_agents);
         if(agent.isAvailable()){
            int skill = agent.getSkills();
            String skillKey = String.valueOf(skill);
            int count = 1;
            if(skillHistogram.containsKey(skillKey)){
               int countSkill = (Integer)skillHistogram.get(skillKey);
               count += countSkill;
            }//if
            skillHistogram.put(skillKey, count);
         }//if
      }//for

      return skillHistogram;
   }//getAvailableAgentSkillHistogram

   /* This function creates a list of the agents that possess each skill,
*/
   /*   where the list is ordered by the number of agents possessing a skill
*/
   /*   in ascending order
*/
   public SkillPriorityList getAgentSkillPriorityList(){
      SkillPriorityList agentSkillPriorityList = new SkillPriorityList();
      for(int i = 0; i < getLength(m_agents); i++){
         AgentX agent = (AgentX)getItem(i,m_agents);
         int skill = agent.getSkills();
         SkillPriorityItem skillItem;
                              /*
*/
         if(agentSkillPriorityList.contains(skill)){
            skillItem = agentSkillPriorityList.remove(skill);
         }else{
            skillItem = new SkillPriorityItem(skill);
         }//else
         skillItem.addAgent(i);
         agentSkillPriorityList.put(skill, skillItem);
      }//for
      return agentSkillPriorityList;
   }//getAgentSkillPriorityList

   /* This function computes the optimal tasks completed
*/
   public void computeOptimal(){
      // If the skill priority list hasn't been created
      if(m_firstSkillPriorityList){
         // Create the skill priority list
         m_agentSkillPriorityList = getAgentSkillPriorityList();
         m_firstSkillPriorityList = false;
      }//if

      Vector<AONTask> currentTasks = new Vector();

      // Loop through the tasks that are not expired or completed
      for(int i = m_tasks.getFirstCurrentIndex(); i < m_tasks.getLength();
i++){
         // Get a copy of the task
         AONTask task = new AONTask((AONTask)m_tasks.getItem(i));
         // If the task has not been used to compute the optimal performance
```

```
         // and the task is not impossible
         if(!m_tasksUsedComputeOptimal.contains(i)
            && !task.taskImpossible(m_agentSkillPriorityList)){
            currentTasks.add(task);
            Vector<Integer> skills = task.getSkillsRequired();
            // Loop through the skills the current task requires
            for(int j = 0; j < skills.size(); j++){
               int skill = skills.elementAt(j);
               SkillPriorityItem item = m_agentSkillPriorityList.get(skill);
               int agentID = item.getAvailableAgent();
               while(task.isNeeded(skill) && agentID >= 0){
                  task.addAgentCommitted(agentID,task.findSkillIndex(skill));
                  item.setAgentTimeLeft(agentID,task.getDuration());
                  agentID = item.getAvailableAgent();
               }//while
            }//for
            m_tasksUsedComputeOptimal.add(i);
         }//if
      }//for

      int totalFilled = 0, total = 0;
      for(int i = 0; i < currentTasks.size(); i++){
         AONTask task = (AONTask)getItem(i,currentTasks);
         if(task.agentsNeeded() == 0){
            m_numberOfOptimalTeamsFormed++;
         }//if
      }//for
      m_optimalPerformance =
(float)m_numberOfOptimalTeamsFormed/m_numberOfTasksIntroduced;
   }//computeOptimal

   /* This function saves several screen captures of structural agents
*/
   /*   during the steps of a simple rewiring scenario
*/
   public void printStructuralAgents(){
      System.out.println("************************************");
      System.out.println("*          Print Structural Agents  *");
      System.out.println("************************************");

      m_agents.clear();

      int x[] = {20,  20, 50, 110, 110, 140, 140};
      int y[] = {40, 120, 80,  10,  80,  40, 120};
      String type = "Structural";
      for(int i = 0; i < 7; i++){
         AgentX agent = addAgent(i, x[i], y[i], type);
         initializeAgent(agent);
      }//for

      /***************
       * 0      3
       *             5
       *   2    4
       *
       * 1          6
       */

      int newNeighbor;       /*
*/

      AgentX a0 = (AgentX)getItem(0,m_agents);
      AgentX a1 = (AgentX)getItem(1,m_agents);
```

```
      AgentX a2 = (AgentX)getItem(2,m_agents);
      AgentX a3 = (AgentX)getItem(3,m_agents);
      AgentX a4 = (AgentX)getItem(4,m_agents);
      AgentX a5 = (AgentX)getItem(5,m_agents);
      AgentX a6 = (AgentX)getItem(6,m_agents);

      a0.addNeighbor(2);
      a1.addNeighbor(2);
      a2.addNeighbor(4);
      a3.addNeighbor(4);
      a4.addNeighbor(5);
      a6.addNeighbor(4);

      m_drawStatusText = false;
      performScreenCapture("screenshots/StructuralAgents_00.jpg",200,220);

      incrementAllAgentConnections(6);
      performScreenCapture("screenshots/StructuralAgents_01.jpg",200,220);

      incrementAllAgentConnections(5);
      performScreenCapture("screenshots/StructuralAgents_02.jpg",200,220);

      newNeighbor = a0.rewire();
      performScreenCapture("screenshots/StructuralAgents_03.jpg",200,220);
   }//printStructuralAgents

   /* This function saves several screen captures of performance agents
*/
   /*   during the steps of a simple rewiring scenario
*/
   public void printPerformanceAgents(){
      System.out.println("************************************");
      System.out.println("*          Print Performance Agents *");
      System.out.println("************************************");

      m_agents.clear();

      int x[] = {20,  20, 50, 110, 110, 140, 140};
      int y[] = {40, 120, 80,  10,  80,  40, 120};
      String type = "Performance";
      for(int i = 0; i < 7; i++){
         AgentX agent = addAgent(i, x[i], y[i], type);
         initializeAgent(agent);
      }//for

      /***************
       * 0      3
       *            5
       *   2    4
       *
       * 1          6
       */

      int newNeighbor;       /*
*/

      AgentX a0 = (AgentX)getItem(0,m_agents);
      AgentX a1 = (AgentX)getItem(1,m_agents);
      AgentX a2 = (AgentX)getItem(2,m_agents);
      AgentX a3 = (AgentX)getItem(3,m_agents);
      AgentX a4 = (AgentX)getItem(4,m_agents);
      AgentX a5 = (AgentX)getItem(5,m_agents);
      AgentX a6 = (AgentX)getItem(6,m_agents);
```

```
      a0.setTeamsJoined(10);
      a0.setSuccessfulTeamsFormed(5);
      a1.setTeamsJoined(10);
      a1.setSuccessfulTeamsFormed(5);
      a2.setTeamsJoined(10);
      a2.setSuccessfulTeamsFormed(5);
      a3.setTeamsJoined(10);
      a3.setSuccessfulTeamsFormed(5);
      a4.setTeamsJoined(10);
      a4.setSuccessfulTeamsFormed(5);
      a5.setTeamsJoined(10);
      a5.setSuccessfulTeamsFormed(5);
      a6.setTeamsJoined(10);
      a6.setSuccessfulTeamsFormed(5);

      a0.addNeighbor(2);
      a1.addNeighbor(2);
      a2.addNeighbor(4);
      a3.addNeighbor(4);
      a4.addNeighbor(5);
      a6.addNeighbor(4);

      m_drawStatusText = false;
      performScreenCapture("screenshots/PerformanceAgents_00.jpg",200,220);

      incrementAllAgentConnections(6);
      performScreenCapture("screenshots/PerformanceAgents_01.jpg",200,220);

      incrementAllAgentConnections(5);
      performScreenCapture("screenshots/PerformanceAgents_02.jpg",200,220);

      newNeighbor = a0.rewire();
      performScreenCapture("screenshots/PerformanceAgents_03.jpg",200,220);
   }//printPerformanceAgents

   /* This function saves several screen captures of inventory agents
*/
   /*   during the steps of a simple rewiring scenario
*/
   public void printInventoryAgents(){
      System.out.println("************************************");
      System.out.println("*          Print Inventory Agents   *");
      System.out.println("************************************");

      m_agents.clear();

      int x[] = {20,  20, 50, 110, 110, 140, 140};
      int y[] = {40, 120, 80,  10,  80,  40, 120};
      String type = "Inventory";
      for(int i = 0; i < 7; i++){
         AgentX agent = addAgent(i, x[i], y[i], type);
         initializeAgent(agent);
      }//for

      /***************
       * 0      3
       *             5
       *   2    4
       *
       * 1          6
       */
```

```
      int newNeighbor;       /*
*/

      AgentX a0 = (AgentX)getItem(0,m_agents);
      AgentX a1 = (AgentX)getItem(1,m_agents);
      AgentX a2 = (AgentX)getItem(2,m_agents);
      AgentX a3 = (AgentX)getItem(3,m_agents);
      AgentX a4 = (AgentX)getItem(4,m_agents);
      AgentX a5 = (AgentX)getItem(5,m_agents);
      AgentX a6 = (AgentX)getItem(6,m_agents);

      a0.addSkill(1);
      a1.addSkill(8);
      a2.addSkill(4);
      a3.addSkill(3);
      a5.addSkill(7);
      a4.addSkill(5);
      a6.addSkill(4);

      a0.addNeighbor(2);
      a1.addNeighbor(2);
      a2.addNeighbor(4);
      a3.addNeighbor(4);
      a4.addNeighbor(5);
      a6.addNeighbor(4);

      Vector<Integer>failedSkillCount = new Vector();
      for(int i = 0; i < 10; i++){
         failedSkillCount.add(0);
      }//for
      failedSkillCount.setElementAt(49,1);
      failedSkillCount.setElementAt(50,2);
      a4.setFailedSkillCount(failedSkillCount);

      m_drawStatusText = false;
      performScreenCapture("screenshots/InventoryAgents_00.jpg",200,220);

      incrementAllAgentConnections(6);
      performScreenCapture("screenshots/InventoryAgents_01.jpg",200,220);

      incrementAllAgentConnections(5);
      performScreenCapture("screenshots/InventoryAgents_02.jpg",200,220);

      a4.update(m_tasks);
      newNeighbor = a4.rewire();
      performScreenCapture("screenshots/InventoryAgents_03.jpg",200,220);

      a4.resetNeighborhoodStats();
      a4.update(m_tasks);
   }//printInventoryAgents

   /* This function saves several screen captures of egalitarian agents
*/
   /*   during the steps of a simple rewiring scenario
*/
   public void printEgalitarianAgents(){
      System.out.println("************************************");
      System.out.println("*          Print Egalitarian Agents *");
      System.out.println("************************************");

      m_agents.clear();

      int x[] = {20,  20, 50, 110, 110, 140, 140};
```

```
      int y[] = {40, 120, 80,  10,  80,  40, 120};
      String type = "Egalitarian";
      for(int i = 0; i < 7; i++){
         AgentX agent = addAgent(i, x[i], y[i], type);
         initializeAgent(agent);
      }//for

      /***************
       * 0      3
       *            5
       *   2    4
       *
       * 1          6
       */

      int newNeighbor;       /*
*/

      AgentX a0 = (AgentX)getItem(0,m_agents);
      AgentX a1 = (AgentX)getItem(1,m_agents);
      AgentX a2 = (AgentX)getItem(2,m_agents);
      AgentX a3 = (AgentX)getItem(3,m_agents);
      AgentX a4 = (AgentX)getItem(4,m_agents);
      AgentX a5 = (AgentX)getItem(5,m_agents);
      AgentX a6 = (AgentX)getItem(6,m_agents);

      a0.addNeighbor(2);
      a1.addNeighbor(2);
      a2.addNeighbor(4);
      a3.addNeighbor(4);
      a4.addNeighbor(5);
      a6.addNeighbor(4);

      m_drawStatusText = false;
      performScreenCapture("screenshots/EgalitarianAgents_00.jpg",200,220);

      incrementAllAgentConnections(6);
      performScreenCapture("screenshots/EgalitarianAgents_01.jpg",200,220);

      incrementAllAgentConnections(5);
      performScreenCapture("screenshots/EgalitarianAgents_02.jpg",200,220);

      newNeighbor = a0.rewire();
      performScreenCapture("screenshots/EgalitarianAgents_03.jpg",200,220);
   }//printEgalitarianAgents

   /* This function saves several screen captures illustrating how
*/
   /*   communication depth works when gathering rewiring information
*/
   public void printCommunicationDepth(){
      System.out.println("*************************************");
      System.out.println("*          Print Communication Depth *");
      System.out.println("*************************************");

      m_agents.clear();

      int x[] = {20,  20, 50, 110, 110, 140, 140};
      int y[] = {40, 120, 80,  10,  80,  40, 120};
      String type = "Structural";
      for(int i = 0; i < 7; i++){
         AgentX agent = addAgent(i, x[i], y[i], type);
         initializeAgent(agent);
```

```
        }//for

        /***************
         * 0       3
         *             5
         *   2     4
         *
         * 1           6
         */

        int newNeighbor;       /*
*/

        AgentX a0 = (AgentX)getItem(0,m_agents);
        AgentX a1 = (AgentX)getItem(1,m_agents);
        AgentX a2 = (AgentX)getItem(2,m_agents);
        AgentX a3 = (AgentX)getItem(3,m_agents);
        AgentX a4 = (AgentX)getItem(4,m_agents);
        AgentX a5 = (AgentX)getItem(5,m_agents);
        AgentX a6 = (AgentX)getItem(6,m_agents);

        a0.addNeighbor(2);
        a1.addNeighbor(2);
        a2.addNeighbor(4);
        a3.addNeighbor(4);
        a4.addNeighbor(5);
        a6.addNeighbor(4);

        a0.setState(10);
        a1.setState(10);
        a2.setState(10);
        a3.setState(10);
        a4.setState(10);
        a5.setState(10);
        a6.setState(10);

        Color darkOrange = new Color(170,120,30);
        Color mediumOrange = new Color(230,120,30);
        Color yellow = Color.yellow;

        a0.setColor(Color.red);
        a1.setColor(Color.lightGray);
        a2.setColor(Color.lightGray);
        a3.setColor(Color.lightGray);
        a4.setColor(Color.lightGray);
        a5.setColor(Color.lightGray);
        a6.setColor(Color.lightGray);

        incrementAllAgentConnections(11);

        m_drawStatusText = false;
        performScreenCapture("screenshots/CommunicationDepth_00.jpg",200,220);

        a2.setColor(darkOrange);

        performScreenCapture("screenshots/CommunicationDepth_01.jpg",200,220);

        a1.setColor(mediumOrange);
        a4.setColor(mediumOrange);

        performScreenCapture("screenshots/CommunicationDepth_02.jpg",200,220);

        a3.setColor(yellow);
```

```
      a5.setColor(yellow);
      a6.setColor(yellow);

      performScreenCapture("screenshots/CommunicationDepth_03.jpg",200,220);
   }//printCommunicationDepth

   /* This function is used to artificially age new network connections in
*/
   /*   some of the unit tests for TestAON
*/
   private void incrementAllAgentConnections(int count){
      for(int i = 0; i < m_agents.size(); i++){
         AgentX agent = (AgentX)getItem(i,m_agents);
         for(int j = 0; j < count; j++){
            agent.incrementNeighborConnections();
         }//for
      }//for
   }//incrementAllAgentConnections

   /* This function pauses execution waiting for input from the console
*/
   public void pauseTest(String message){
      try{
         char input = ' ';
         byte temp[] = new byte[1];
         System.out.print("\n" + message + "\nPress <enter> to continue...");
         while(input != '\n'){
            System.in.read(temp);
            input = (char)temp[0];
         }//while
      }catch(IOException ex){
      }//catch
   }//pauseTest
}//class TestAON

-------------------------------TestAON.java-------------------------------------
-

-------------------------------UniformAgent.java--------------------------------
-

import java.util.*;

/**
 * UniformAgent CLASS
 *   This is an agent that seeks to rewire to agents with the same skills
 *     the agent possesses
 */
public class UniformAgent extends AgentX{
   /* constructor
*/
   public UniformAgent(TestAON p,int id, int x, int y, Vector agents){
      super(p,id,x,y,"Uniform",agents);
   }//constructor

   /* This function is used to find the best agent recommendation for a new
*/
   /*   neighbor by doing a recursive search of the network until depth has
*/
   /*   reached 0. The best agent recommendation is returned back to the
*/
   /*   agent who originated the request for agent recommendations
*/
```

```
   public AgentRecommendation recommendNeighbor(int depth, Vector
agentsVisited, AgentX sourceAgent){
      int bestAgentID = -1;
      float bestCompareValue = 999999.0f;
      depth--;
      agentsVisited.add(m_id);
      if(depth <= 0){
         if(!sourceAgent.hasNeighbor(m_id) && m_id != sourceAgent.getID()){
            bestAgentID = m_id;
            bestCompareValue = getSkillComparison(sourceAgent.getSkills());
         }//if
      }else{
         for(int i = 0; i < m_neighbors.getLength(); i++){
            int id = m_neighbors.getItem(i).agentID();
            if(!contains(id,agentsVisited)){
               AgentX neighbor = (AgentX)getAgent(id);
               AgentRecommendation agentRecommendation =
neighbor.recommendNeighbor(depth,agentsVisited,sourceAgent);
               if(agentRecommendation.getID() >= 0){
                  int agentDegree =
getAgent(agentRecommendation.getID()).getDegree();
                  if((int)agentRecommendation.getValue() < bestCompareValue &&
agentDegree < m_maxDegree){
                     bestAgentID = agentRecommendation.getID();
                     bestCompareValue = agentRecommendation.getValue();
                  }//if
               }//if
            }//if
         }//for
         if(!sourceAgent.hasNeighbor(m_id) && m_id != sourceAgent.getID()){
            if(bestAgentID == -1){
               bestAgentID = m_id;
               bestCompareValue = getSkillComparison(sourceAgent.getSkills());
            }else{
               float personalComparison =
getSkillComparison(sourceAgent.getSkills());
               if(personalComparison < bestCompareValue){
                  bestAgentID = m_id;
                  bestCompareValue = personalComparison;
               }//if
            }//else
         }//if
      }//else
      return new AgentRecommendation(bestAgentID,bestCompareValue);
   }//recommendNeighbor

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "UniformAgent\n" + super.toString() + "\n";
      return value;
   }//toString
}//class UniformAgent

------------------------------UniformAgent.java---------------------------------
-

-------------------------UniformImpatientAgent.java----------------------------
-

import java.util.*;

/**
```

```
 * UniformImpatientAgent CLASS
 *   This is an agent that seeks to rewire to agents with the same skills
 *     the agent possesses
 *   -This agent is 'impatient' because it builds off the Egalitarian agent
 *     by choosing to drop its commitment to a task when the skills left to be
 *     filled in the task exceed the number of neighboring agents who are not
 *     committed to a task
 */
public class UniformImpatientAgent extends UniformAgent {
   /* constructor
*/
   public UniformImpatientAgent(TestAON p,int id, int x, int y, Vector agents){
      super(p,id,x,y,agents);
      m_type = "UniformImpatient";
   }//constructor

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      boolean dropTask = false;
      if(getUnCommittedNeighborCount() < task.unFilledSkillsCount()){
         dropTask = true;
      }//if
      return dropTask;
   }//chooseToDropTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "UniformImpatientAgent\n" + super.toString() + "\n";
      return value;
   }//toString
}//class UniformImpatientAgent

-------------------------UniformImpatientAgent.java-----------------------------
-

---------------------------UniformStratAgent.java-----------------------------
-

import java.util.*;

/**
 * UniformStratAgent CLASS
 *   This is an agent that seeks to rewire to agents with the same skills
 *     the agent possesses
 *   -This agent is 'strategic' because it builds off the Egalitarian agent
 *    by choosing tasks to join and tasks to propose that are the best match
 *    for the skills possessed by the agent's uncommitted neighbors
 */
public class UniformStratAgent extends UniformAgent {
   /* constructor
*/
   public UniformStratAgent(TestAON p,int id, int x, int y, Vector agents){
      super(p,id,x,y,agents);
      m_type = "UniformStrat";
   }//constructor

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
```

```
       m_edgesChanged = 0;
       writeLog(m_id + " updateUncommitted:");
       // Look at all tasks which your immediate neighbor has committed to
       // Select the one which is best based on (1) one with more committed
agents
       // (2) one for which your neighbors have the most chance of satisfying
       AONTask taskJoin = pickNeighborTask(tasks);
       if(taskJoin != null){
          acceptTask(taskJoin);
          m_state = C_COMMITTED;
          m_numberOfTeamsJoined++;
          m_committedTimeRemaining = m_maxCommittedTime;
          m_lastWorkCatagory = C_JOINED;
         return;
      }//if

      float random = rand.nextFloat();
      // WaitPercent can be fixed or dynamic (based on success with current
strategy)
      Boolean wantToWait = (random < m_waitPercent);
      if(chooseToAdapt()){
         rewire();
      }else if(wantToWait){
         return;
      }else{
         //Propose a new task
         // Look at all available tasks which you can do.
         // Select the one which is best based on (1) random choice
         // (2) one which your neighbors have the most chance of satisfying
         AONTask task = pickTask(tasks);
         if(task != null){
            acceptTask(task);
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
            m_lastWorkCatagory = C_PROPOSED;
            m_numberOfTeamsJoined++;
         }//if
      }//else
   }//updateUncommitted

   /* This function returns the best task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickTask(AONTaskList tasks){
      return pickBestTask(tasks);
   }//pickTask

   public AONTask pickNeighborTask(AONTaskList tasks){
      return pickBestNeighborTask(tasks);
   }//pickNeighborTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "UniformStratAgent\n" + super.toString() + "\n";
      return value;
   }//toString
}//class UniformStratAgent

--------------------------UniformStratAgent.java------------------------------
-
```

```
----------------------UniformStratImpatientAgent.java-------------------------
-

import java.util.*;

/**
 * UniformStratImpatientAgent CLASS
 *   This is an agent that seeks to rewire to agents with the same skills
 *     the agent possesses
 *   -This agent is 'strategic' because it builds off the Egalitarian agent
 *    by choosing tasks to join and tasks to propose that are the best match
 *    for the skills possessed by the agent's uncommitted neighbors
 *   -This agent is 'impatient' because it builds off the Egalitarian agent
 *     by choosing to drop its commitment to a task when the skills left to be
 *     filled in the task exceed the number of neighboring agents who are not
 *     committed to a task
 */
public class UniformStratImpatientAgent extends UniformAgent {
   /* constructor
*/
   public UniformStratImpatientAgent(TestAON p,int id, int x, int y, Vector
agents){
      super(p,id,x,y,agents);
      m_type = "UniformStratImpatient";
   }//constructor

   /* This function updates the agent when its state is uncommitted
*/
   public void updateUncommitted(AONTaskList tasks){
       m_edgesChanged = 0;
       writeLog(m_id + " updateUncommitted:");
       // Look at all tasks which your immediate neighbor has committed to
       // Select the one which is best based on (1) one with more committed
agents
       // (2) one for which your neighbors have the most chance of satisfying
       AONTask taskJoin = pickNeighborTask(tasks);
       if(taskJoin != null){
          acceptTask(taskJoin);
          m_state = C_COMMITTED;
          m_numberOfTeamsJoined++;
          m_committedTimeRemaining = m_maxCommittedTime;
          m_lastWorkCatagory = C_JOINED;
         return;
      }//if

      float random = rand.nextFloat();
      // WaitPercent can be fixed or dynamic (based on success with current
strategy)
      Boolean wantToWait = (random < m_waitPercent);
      if(chooseToAdapt()){
         rewire();
      }else if(wantToWait){
         return;
      }else{
         //Propose a new task
         // Look at all available tasks which you can do.
         // Select the one which is best based on (1) random choice
         // (2) one which your neighbors have the most chance of satisfying
         AONTask task = pickTask(tasks);
         if(task != null){
            acceptTask(task);
            m_state = C_COMMITTED;
            m_committedTimeRemaining = m_maxCommittedTime;
```

```
            m_lastWorkCatagory = C_PROPOSED;
            m_numberOfTeamsJoined++;
         }//if
      }//else
   }//updateUncommitted

   /* This function returns the best task a neighbor has committed to for
*/
   /*   which this agent is needed
*/
   public AONTask pickNeighborTask(AONTaskList tasks){
      return pickBestNeighborTask(tasks);
   }//pickNeighborTask

   /* This function returns the best task which this agent is needed for
*/
   public AONTask pickTask(AONTaskList tasks){
      return pickBestTask(tasks);
   }//pickTask

   /* This function returns whether the agent has chosen to drop its
*/
   /*   commitment to the task it is currently committed to
*/
   public boolean chooseToDropTask(AONTask task){
      boolean dropTask = false;
      if(getUnCommittedNeighborCount() < task.unFilledSkillsCount()){
         dropTask = true;
      }//if
      return dropTask;
   }//chooseToDropTask

   /* This function returns a string representation of the agent
*/
   public String toString(){
      String value = "UniformStratImpatientAgent\n" + super.toString() + "\n";
      return value;
   }//toString
}//class UniformStratImpatientAgent

----------------------UniformStratImpatientAgent.java-------------------------
-

-----------------------------XMLElement.java-----------------------------------
-

import java.util.*;

/**
 * XMLElement Class
 *
 *
 */
public class XMLElement{
   private String name;      /* The name of the XML element
*/
   private XMLElement parent;/* The XML element that is the XML element's
*/
                            /*   parent
*/

   private Hashtable attributes;
```

```
                              /* The attributes for the XML element stored by
*/
                              /*   key and value with attribute name as key
*/
   private Vector attributeNames;
                              /* The names of the XML element's attributes
*/

   private Hashtable elements;
                              /* The child XML elements for the XML element
*/
                              /*   stored by key and value where the element's
*/
                              /*   name is the key
*/
   private Vector elementNames;
                              /* The names of the child XML elements
*/

   /* constructor
*/
   public XMLElement(){
      attributes = new Hashtable();
      attributeNames = new Vector();
      elements = new Hashtable();
      elementNames = new Vector();
   }//constructor

   /* constructor
*/
   public XMLElement(String n, XMLElement p){
      name = n;
      parent = p;
      attributes = new Hashtable();
      attributeNames = new Vector();
      elements = new Hashtable();
      elementNames = new Vector();
   }//constructor

   /* This function returns true if the specified item is found in the array
*/
   /*   or false if the specified item is not in the array
*/
   public boolean contains(Object item, Vector array){
      return array.contains(item);
   }//contains

   /* This function returns the item in the array at the specified index
*/
   public Object getItem(int index, Vector array){
      return array.elementAt(index);
   }//getItem

   /* This function removes the item at the specified index in the array
*/
   public Object removeItem(int index, Vector array){
      return array.remove(index);
   }//removeItem

   /* This function sets the item in the specified index in the array to be
*/
   /*   the specified value
*/
```

```
   public void setItem(int index, Vector array, Object value){
      array.set(index,value);
   }//setItem

   /* This function returns the number of items in the array
*/
   public int getLength(Vector array){
      return array.size();
   }//getArrayLength

   /* This function returns the name of this XML element
*/
   public String getName(){
      return name;
   }//getName

   /* This function adds an attribute for this XML element using the
*/
   /*   attribute name and value specified
*/
   public void addAttribute(String name, String value){
      if(attributeNames.add(name)){
         attributes.put(name,value);
      }//if
   }//addAttribute

   /* This function returns the value of an attribute for this XML element
*/
   /*   by using the specified index into the attribute list
*/
   public String getAttribute(int index){
      if(getLength(attributeNames) > 0 && index < getLength(attributeNames)){
         return (String)attributes.get((String)getItem(index,attributeNames));
      }else{
         return "";
      }//else
   }//getAttribute

   /* This function returns the name of an attribute for this XML element
*/
   /*   by using the specified index into the attribute list
*/
   public String getAttributeName(int index){
      if(getLength(attributeNames) > 0 && index < getLength(attributeNames)){
         return (String)getItem(index,attributeNames);
      }else{
         return "";
      }//else
   }//getAttribute

   /* This function returns the value of an attribute for this XML element
*/
   /*   by using the specified attribute name
*/
   public String getAttribute(String name){
      return (String)attributes.get(name);
   }//getAttribute

   /* This function returns the number of attributes for this XML element
*/
   public int getNumAttributes(){
      return getLength(attributeNames);
   }//getNumAttributes
```

```
   /* This function adds a child XML element for this XML element using the
*/
   /*   specified child XML element name and child XML element object
*/
   public void addElement(String name, XMLElement item){
      if(elementNames.add(name)){
         elements.put(name,item);
      }//if
   }//addElement

   /* This function returns a child XML element for this XML element by using
*/
   /*   the specified index into the child XML element list
*/
   public XMLElement getElement(int index){
      if(getLength(elementNames) > 0 && index < getLength(elementNames)){
         return (XMLElement)elements.get(getItem(index,elementNames));
      }else{
         return null;
        }//else
   }//getElement

   /* This function returns the XML element corresponding to the specified
*/
   /*   name from the list of child XML elements this XML element has
*/
   public XMLElement getElement(String name){
      return (XMLElement)elements.get(name);
   }//getElement

   /* This function returns the number of child XML elements this XML element
*/
   /*   has
*/
   public int getNumElements(){
      return getLength(elementNames);
   }//getNumElements

   /* This function returns the XML element that is the parent to this
*/
   /*   XML element
*/
   public XMLElement getParent(){
      return parent;
   }//getParent

   /* This function sets the parent XML element for this XML element
*/
   /*   to be the specified XML element
*/
   public void setParent(XMLElement p){
      parent = p;
   }//setParent

   /* This function deletes the child XML element specified by the index into
*/
   /*   the list of XML element children this XML element has
*/
   public void deleteElement(int index){
      String name = (String)getItem(index,elementNames);
      elements.remove(name);
      elementNames.remove(index);
```

```
   }//deleteElement

   /* This function removes all child XML elements for this XML element
*/
   public void deleteElements(){
      elements.clear();
      elementNames.clear();
   }//deleteElements
}//XMLElement

------------------------------XMLElement.java-----------------------------------
-
```