6-14-2018

Effects of Dynamic Goals on Agent Performance

Nathan R. Ball

Effects of Dynamic Goals on Agent Performance

THESIS

Nathan R. Ball, 2d Lt, USAF

AFIT-ENG-MS-18-J-003

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-18-J-003

EFFECTS OF DYNAMIC GOALS ON AGENT PERFORMANCE

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Nathan R. Ball, B.S.E.E.

2d Lt, USAF

May 24, 2018

AFIT-ENG-MS-18-J-003

EFFECTS OF DYNAMIC GOALS ON AGENT PERFORMANCE

THESIS

Nathan R. Ball, B.S.E.E.
2d Lt, USAF

Committee Membership:

Maj Jason Bindewald, PhD
Chair

Dr. Gilbert Peterson
Member

Dr. Micheal Miller
Member

AFIT-ENG-MS-18-J-003

Abstract

Autonomous systems are increasingly being used for complex tasks in dynamic environments. Robust automation needs to be able to establish it's current goal and determine when the goal has changed. In human-machine teams autonomous goal detection is an important component of maintaining shared situational awareness between both parties. This research investigates how different categories of goals affect autonomous change detection in a dynamic environment.

In order to accomplish this goal, a set of autonomous agents were developed to perform within an environment with multiple possible goals. The agents perform the environmental task while monitoring for goal changes. The experiment tests the agents over a range of goal changes to determine how detection performance is affected by the different categories of goals.

Results show that detection is highly dependent on what goal is being switch to and from. The point similarity between goals is the most significant factor in evaluating the change detection time. An additional experiment improved upon the goal agent and demonstrated the importance of having the proper perception mechanics for feedback within the environment.

Table of Contents

Page

Page

List of Figures

List of Tables

EFFECTS OF DYNAMIC GOALS
ON AGENT PERFORMANCE

I. Introduction

1.1 Department of Defense Motivation

Autonomous agents continue to expand into increasingly complex domains where greater capabilities are required to achieve mission effectiveness. Whether autonomous agents are used to fly a drone or detect objects within surveillance images, the mission goal will not always remain constant. An important capability for future autonomous agents will be their ability to detect changes in their goal structure and to adapt to changes forcing the agent to switch to a new goal. The ability to intelligently determine its goal based on environmental factors is a vital capability for both fully autonomous systems and agents within human machine teams.

Within fully autonomous systems, goal determination is necessary for any system operating in a highly dynamic environment. For example, if an autonomous supply convoy detects an improvised explosive device on the road, it should change its goal to ensure that proper authorities are alerted and other vehicles are aware of the device's location. In a fully autonomous system, this action should be taken without a human having to interact with the automation. As humans get pulled farther outside the control loop, the more important goal determination becomes.

For human machine teams, goal recognition is equally important. In 2015 the United Sates Air Force Chief Scientist released *Autonomous Horizons* [6], a document detailing a roadmap for the future of autonomy in the service. One of the key

Figure 1. Model of shared situational awerness, from Endsley et al. [6].

objectives is utilizing the potential benefits of the human-machine teams to increase mission effectiveness. One of the key issues to achieving this symbiosis is a shared situational awareness between the human and autonomous agent. Figure 1 shows the various levels of the shared situation awareness model. At the top of the model is goal alignment between the human and machine. Without goal alignment, the machine and human will be working to complete potentially conflicting goals. For example, in an airplane there will be conflict if an agent is attempting to land while the pilot is instead planning to circle around runway [6]. Goal detection provides agents with the flexibility to react to changes within the environment and maintain alignment with their human teammate.

Having a shared goal is essential for a high functioning team to ensure that all members are taking actions that help to achieve the same goal. By definition, a team is a group that has come together to achieve a common goal. In human machine teams shared goals are just as important to ensure that the human is not fighting the automation and showing that the human can trust the automation. Goal flexibility is one way to increase human trust in the automation, further increasing team

performance. A human will trust automation more as it demonstrates that it shares the same goal as the human [14]. Increased trust has the added benefit of allowing the human to focus on their work for longer, as opposed to continuously monitoring the work the automation is doing [16]. This added focus fosters the notion that the human and automation are truly a team.

1.2 Research Motivation

Game environments provide a great way to test methods of goal detection within dynamic environments that can be easily modified. This thesis examines the problem of goal detection using a gameplay environment. Goal determination and adaption is not an inherently new concept in games. Many games have implemented an adaptive artificial intelligence that can reason over goals to improve realism and difficulty. For example, the artificial intelligence enemies in, *No One Lives Forever 2: A Spy in H.A.R.M.s Way* [22] and *F.E.A.R.* [23], dynamically choose between different goals and create an action plan according to the current goal. In both games, however, the agents are hard coded with all the possible goal states and need only determine which goal it should accomplish. They do not formulate new goals beyond the ones that have been hard coded. In this research the agent has no knowledge of the possible goal changes and must observe the environment to determine the current goal state.

The environment utilized for this research is a dynamic tablet based video game that is played by a collection of agents, in designed experiments, the goal changes mid-play, forcing the adaptive agent to detect and adapt to the new goal to maximize its final score. The environment utilized for this research is comparable to a real-world air traffic control environment, where a human controller monitors aircraft takeoff and landings. In real-world air traffic control, the goal for aircraft can suddenly shift due to the changes within the environment. An aircraft originally told to go to a

runway may need to do another pass around due to unforeseen circumstances. This unpredictability forces controllers to stay attentive to changes in the environment.

The notions of concept drift and reinforcement learning are leveraged to design the adaptive learning agent. For this research an adaptive learning agent will similarly need to monitor the environment for potential changes and adapt its action accordingly when a change is detected.

1.3 Research Objective

1.3.1 Research Problem.

This research will examines performance of an adaptive agent within the dynamic goal environment. The question this research aims to answer is, "Do different categories of goals affect an agent's ability to detect and adapt to goal changes?" The hypothesis is that different categories of goals affect detection depending on what goals were switched to and from. These differences can be used to inform the types of goals the agent can quickly and accurately detect. This can later be compared to human play to determine differences between how agents and humans detect change.

1.3.2 Investigative Questions.

To answer the proposed research question, the following investigative questions will be explored:

- Investigative Question 1: How can goals be classified within the context of a gameplay environment?

  Hypothesis: Goals can be uniquely classified based on specific gameplay actions the goal requires.

- Investigative Question 2: What methods can be used to detect environment

goals?

Hypothesis: Methods of reinforcement learning can be used to learn what the goal of the environment is.

- Investigative Question 3: How can goal changes be detected within the environment?

  Hypothesis: Concept drift detection can be used to detect goal changes though analysis of the points earned over time.

- Investigative Question 4: Do the goals being switched to or from affect the time taken to detect a change?

  Hypothesis: Specific categories of goals will take longer to detect after being changed to, no matter the starting goal.

1.4 Methodology Overview

The methodology for testing the research question involved running experiments in the *Space Navigator* route creation environment, modified to have multiple goal conditions. Where the game previously had only one goal the player had to accomplish, modifying the game allowed for multiple different goals. Each new goal required the player to alter his or her strategy and play. Next, a set of autonomous agents to play *Space Navigator* under multiple goal conditions were developed. The collection of agents works to detect the current goal condition and take the appropriate actions for that goal. Additionally, the agents track the goal and detect when a goal change has occurred in the environment. The agents played *Space Navigator* over a series of trials in which the goal suddenly changes in the middle of the game. The goals being switched to and from and the goal change time were modified for each trial. Finally, the results from the autonomous play are analyzed to determine which goals were the

most difficult to detect and adapt too. This informs future work that can contrast these results with those of a human to determine the difference between how humans and autonomous agents identify goal changes and what types of goals are easier for each to adapt to.

1.5 Thesis Overview

The remainder of this is arranged as follows. Chapter II introduces a taxonomy of game classification that will be used to define goals, presents the application environment, and explains concept drift and reinforcement learning as it relates to goal detection. Chapter III covers changes made to the application environment for this research, specifics of the agent design, and an overview of the experimental design. Chapter IV investigates the captured results using statistical analysis. Chapter V then presents and analyzes a follow-up experiment based on the results of the first experiment. Chapter VI proposes a human subject experiment to follow-up on this research. Lastly, Chapter VII outlines the major findings of the data to summarize the finding and concludes with a discussion of future work that can be done following this research.

II. Background

This chapter discusses work related to the present research. The chapter first introduces a taxonomy of changes in a gameplay environment, providing a basis for the types of changes that can be made within an environment. Next the application environment, *Space Navigator*, is explained. Next is a brief overview of the A* search algorithm. It is followed by a discussion of concept drift and a generic system for handling drift. Lastly, an overview of reinforcement learning literature is presented, focusing on the concept of exploration versus exploitation and methods for achieving a proper balance between the two. This background forms the foundation for the adaptive agent created for this research.

2.1 Taxonomy of Change in a Gameplay Environment

When examining the effects of dynamic goals on an agent's performance in an environment, it is important to first be able to define the range of goals. What changes to a gameplay environment constitute a goal change? To that end, this research created a taxonomy of change in a gameplay environments. Shown in Figure 1, this taxonomy defines the areas of a game where changes can be made, allowing for better understanding of the effects the change will have. This section defines the elements of this taxonomy and the reasoning behind it's structure.

2.1.1 Elements of a Game.

The first layer of the taxonomy describes the elements of a game; the areas, that when combined, create a gameplay environment. Several researchers have outlined different methods of classifying game elements. Jan Klabber [19] classifies games by their players, rules, and resources. King *et al.* [18] classify games through the use

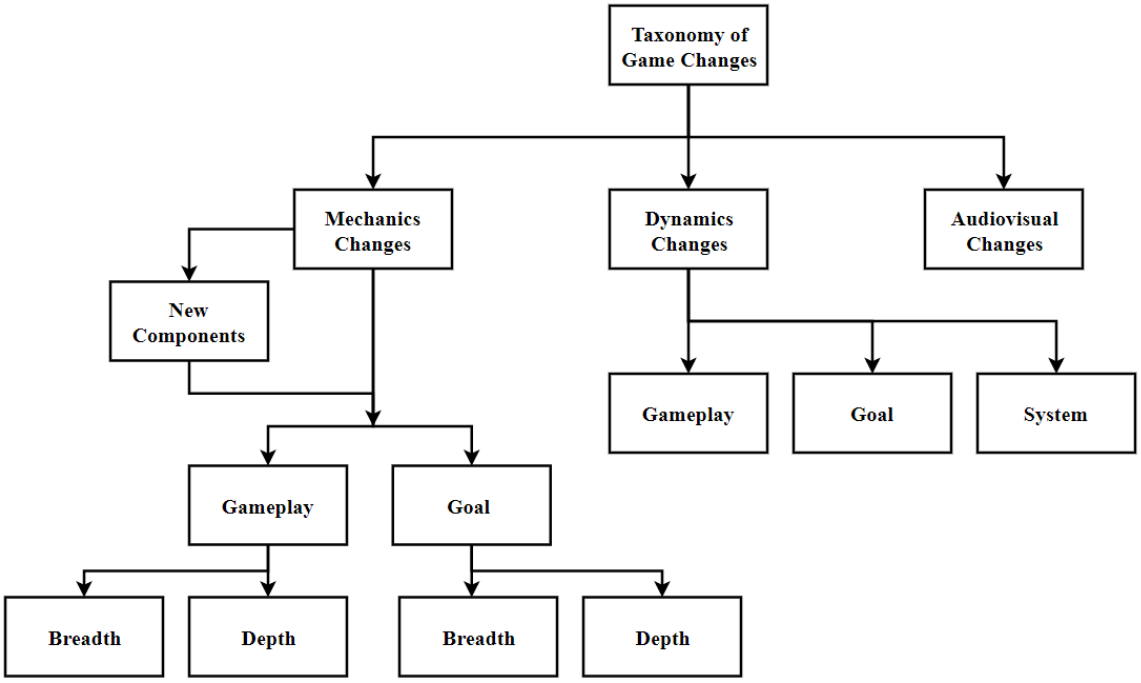

Figure 2. Tree showing the taxonomy of change in a game environment.

of five feature categories: social, manipulation and control, narrative and identity, reward and punishment, and presentation. Salen and Zimmerman [27] define games by their operational, constitutive, and implicit rules; these are the abstract, written, and formal etiquette rules for a game. There is no universally agreed upon framework for defining the elements of a game. The presented taxonomy is modeled after the "Mechanics, Dynamics, and Aesthetics" or MDA model of games by Hunicke *et al.* [15]. This model was chosen because it provides a clear distinction between its elements and can be applied to all games.

According to Hunicke *et al.* [15], mechanics, dynamics, and aesthetics are the three fundamental elements of game design. The model can be used to analyze and decompose games into their fundamental part. The first element of the model, mechanics "describes the particular components of the game" and their rules. The components of *Pac-Man* for example are Pac-Man, the ghosts, Pac-dots, power pellets, bonus fruits, and the game map [21]. These components, together with the rules that define

their operation, make up the game mechanics. Next, dynamics "describes the run-time behavior of the mechanics acting on player inputs and each other outputs over time." Put more simply, dynamics are the interactions between the mechanics, that define the 'play' of the game. For example, in the first-person shooter *Counter-Strike* [31], because the player has low health and does not respawn after death, they play slowly and carefully to avoid being caught off guard.

The last element of the MDA model is aesthetics. In this context aesthetics does not describe the visuals of a game, but rather "describes the desirable emotional responses evoked in the player, when [they] interact with the game system." Hunicke *et al.* [15] list several different aesthetics that games try to elicit in the player such as narrative, challenge, and fellowship. Games can take on multiple different aesthetics to form a cohesive emotional whole. The aesthetics of a game are created through its dynamics. For example, the aesthetic of tension is created in *Counter-Strike* through the slow game speed where enemy encounters are infrequent and resolve quickly.

In the MDA framework, mechanics create dynamics which then create aesthetics. Fundamentally, mechanics are the only part of the framework that can be directly changed, as dynamics and aesthetics are informed through the mechanics. If viewed differently however, numerical and statistical changes to the value of components can be viewed as changes solely to the dynamics of the game. For example, changing the speed of Pac-Man or the ghosts will not change the underlying mechanics of *Pac-Man*. It will only affect the difficulty of the game by altering the players' ability to escape the ghosts; this is a change in the dynamics. In the taxonomy of gameplay change, dynamics will be considered a second category of change in a game, where mechanics are the components in the game and dynamics are the numerical and statistical properties of those components.

The last element of games represented in the taxonomy are the audiovisual ele-

ments of a game. These are how the components of the game physically look and sound. Unlike mechanics and dynamics audiovisual changes do not have any effect on the actual gameplay in the environment. Changing Pac-man to Ms. Pac-man only changes what the player character looks like. These game elements can play a large role in player enjoyment, but they do not affect the rules and interactions that control the components and environment. For this reason, audiovisual changes are not examined further within this taxonomy.

2.1.2 Mechanics.

The next layer represented in the taxonomy describes the changes that can be made to the mechanics of a game. Defining how all the components of a game can change requires mechanics to be decomposed further into separate elements. This is again an area where there is no industry consensus on a formal definition. Due to the countless number of unique gameplay mechanics, even within games of the same genre, generalizing them within a single classification system can be difficult. Most games research does not attempt to classify mechanics, instead simply defining the mechanics as "what the player can do" [25, 28]. Some attempts to classify mechanics, such as that used by Arnab *et al.* [3] and shown in Figure 3, for serious game analysis still cover a wide range of mechanics. While the classification is more defined than most, the mechanics on this list do not all relate directly to gameplay and other mechanics such as 'meta-game' are vague in what they describe.

The present research uses the "Game Bricks" concept created by Djaouti *et al.* [1]. Game bricks were created as a tool used to classify games based upon their mechanics. The game bricks can be used to define the fundamental elements of a game's mechanics. Figure 4 shows the ten bricks outlined in [1]. This method for classifying mechanics was chosen over others because all the mechanics listed

	Game Mechanics		
Behavioral Momentum	Role Play		
Cooperation	Collaboration	Goods / Information	
Selecting / Collecting	Tokens	Cut Scenes / Story	
	Cascading Information	Communal Discovery	
	Questions & Answers	Pareto Optimal	Appointment
Strategy / Planning	Resource Management	Infinite Gameplay	
Capture / Eliminate	Tiles / Grids	Levels	
Game Turns	Action Points	Feedback	
Time pressure	Pavlovian Information	Meta-game	
	Protégé effects	Simulate / Response	Realism
Design / Editing	Movement		
Tutorial	Assessment		
	Competition		
Urgent Optimism	Ownership		
Reward / Penalties	Status	Virality	

Figure 3. List of game mechanics for mapping serious games, recreated from Arnab *et al.* [3].

directly relate to gameplay rules that can be applied directly to components in a gameplay environment. It also provides a clear distinction between input and feedback mechanics. The major limitation of this method is that listed mechanics define only digital single player games. The following list briefly describes the ten mechanics:

1. Avoid: Gameplay of having a component in the environment avoid other components in the environment.

2. Manage: Gameplay of managing some set of resources, such as ammunition in a shooter or gold in a real-time strategy game.

3. Random: Gameplay that contains random elements, such as the role of a die.

4. Shoot: Gameplay of shooting or throwing a component.

5. Create: Gameplay of creating components in a game through player interaction,

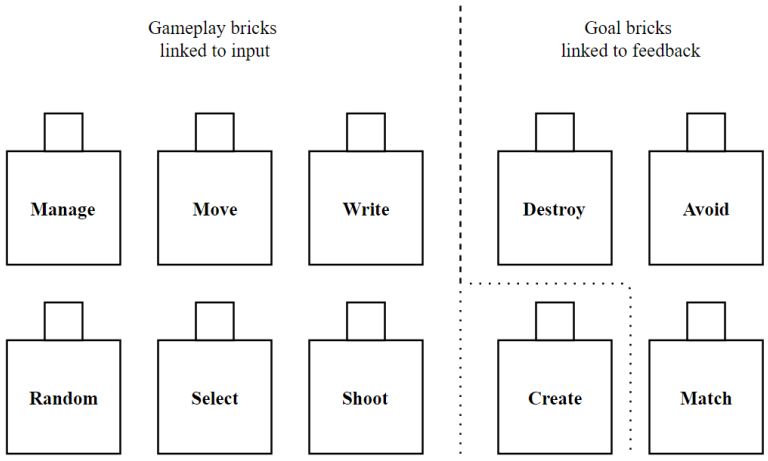

Figure 4. Game bricks divided into play bricks and goal bricks (note that the create brick is a member of both categories), recreated from Djaouti *et al.* [1].

such as crafting items from resources.

6. Destroy: Gameplay where a component is destroyed by another component such as the player destroying the aliens in *Space Invaders* [32].

7. Match: Gameplay of matching the location of components, such as getting to the finish in a racing game.

8. Write: Gameplay of inputting an alphanumeric string into the game.

9. Move: Gameplay of moving a component from one location to another.

10. Select: Gameplay of selecting objects in the environment, such as choosing a piece to move in *Bejeweled* [9].

The gameplay described by these bricks can be combined to describe the overall mechanics of a game. For example, the mechanics of *Pac-Man* [21] can be described using the move, avoid, destroy, and match bricks. The primary gameplay is to *move* Pac-Man around the map. The goal of the game is to *avoid* the ghosts and *destroy* all of the Pac-dots by *matching* Pac-Mans position with the dots.

Within the set of bricks there are two distinct classes of mechanics, as shown in Figure 4. First are 'gameplay' bricks that act on player input into the game. They are the collection of actions that the player is able to do or simply, the actual gameplay of the game. Second are 'goal' bricks that define the goal of the game. These mechanics "observe the game elements and to return an evaluation of the quality of modifications made" [1]. They define what the player is trying to accomplish by providing feedback based on the players' actions. Together these two groups define the mechanics of a game.

Changes to the mechanics of a game involve replacing or adding new bricks to components. New gameplay mechanics can be added through the addition of new gameplay bricks. Goals can be changed through modification of the four goal mechanics. Lastly, as shown in Figure 1, an entirely new component can be added to the game that has its own gameplay and goal mechanics thus adding to both categories.

2.1.3 Implementations of Mechanical Changes.

The final level represented in the taxonomy of gameplay change is concerned with how mechanical changes to the gameplay or goals can be implemented. The game bricks define the two areas of game mechanics a change can affect, but not how the mechanics will be affected by the change. Changes can either add breadth or depth to the mechanics. In his Game Developers Conference (GDC) presentation about formal notation for describing games, designer Raph Koster [20] both defines and creates a method for evaluating the breadth and depth of a game. Koster describes breadth as the number of challenges or actions that can be undertaken in parallel. For example, in a real-time strategy game the player has multiple actions avalible to them at one time, such as creating new buildings, building new units, moving units, and attacking. The more actions available for the player to undertake at once,

the more breadth the game has. Depth is the number of nested actions within a single meta-action. Fighting games, for example, have depth because performing a combination attack requires executing other actions in sequence beforehand. The number of actions required before reaching the final element is the depth.

2.1.4 Dynamics.

Parallel to mechanical changes are dynamics changes. As described earlier, dynamics are the numerical and statistical properties of the game mechanics that affect the balance and feel of the game. Changes to the dynamics will only affect the underlying settings that control the mechanics of the game, not the gameplay or goals available to the player. Dynamics can fall into three different categories of change: gameplay, goal, and system dynamics.

Gameplay and goal dynamics are closely related to their mechanics counterparts. These two groups have game bricks that define them and these bricks have dynamics inherent to them that can be changed. Thus, changes to gameplay dynamics are changes to the settings of the gameplay bricks that the components have. These could be the move speed of a component, how much damage a gun does when it shoots, or how many components can be selected at one time. These are all numerical elements of the gameplay that affect the dynamics. Similarly, goals mechanics are subject to changes to dynamics. The points the player gets for matching locations, time limits to complete the goal, and how much health an enemy has are all examples of the dynamics in goal mechanics. Changes in goal dynamics can incentivize certain goals and change players' behavior.

The last element of dynamics is system dynamics. These are the dynamics that are not related to the gameplay and goals of components in the environment. They do not affect how the components behave or are interacted with, but rather the ‘laws of

nature' and world state of the environment. System dynamics include: the location of objects in the environment, random number generation (RNG) and statistical distributions in the environment, the size of components, and the controls that players use to interact with the environment. None of these dynamics directly affect gameplay or the goal but are inherent to the game system.

2.2 Agent Design

In order to properly design agents, specific architectures must be examined. Several agents will be created to play *Space Navigator* and detect goal changes. Russel & Norvig [26] define an agent as, "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators." This definition can be applied to a wide range of entities, such as humans. In this scenario, however, agents are the computer programs acting in *Space Navigator*. Russel and Norvig list four structures for autonomous agents: simple reflex agents, model-based reflex agents, goal-based agents, and utility based agents.

2.2.1 Simple Reflex Agents.

The simple reflex agent is, as the name suggests, the simplest type of agent. Simple reflex agents select their actions based only on the current world state. This is a simple if-then logic approach. If the world is in state X then do action Y. This type of agent is useful for tasks that do not require any knowledge of past events and can complete their task using only the information currently available to them. The danger of this agent is that they can become stuck in one state, such as a vacuum agent that moves forward, hits a wall, moves back, only to move forward and continue the cycle over again with no way to escape.

2.2.2 Model-Based Reflex Agent.

The model-based reflex agent extends the capabilities of the simple reflex agent by adding an internal state that holds a world history and information on how the agents action have affected the world. This history allows the agent to overcome the infinite loops of the simple reflex agent. By using the internal state knowledge, the agent can predict the outcome of its actions to better inform its choices.

2.2.3 Goal-Based Reflex Agent.

Goal-based reflex agents build on the simple and model-based reflex agents by providing the agent with some goal state it must accomplish. The goal-based agent differs from the reflex agents in the way it selects its actions. Instead of using if-then logic the agent must utilize search and planning to find the actions that complete the goal. Using a search algorithm to find the goal is less efficient than a simple look-up table in the case of a reflex agent. Goal-based agents, however, can be more flexible than reflex agents due to their ability to adapt to new conditions without having decision behavior changed.

2.2.4 Utility-Based Reflex Agent.

The utility agent adds further performance to the goal agent through the addition of a utility function that rates the agent's performance. Because many possible solutions for a goal exist, the utility function provides a means for the agent to decide between actions. The utility function grades each of its possible actions, then performs the one that maximizes its expected reward.

2.2.5 Learning Agent.

Any of the above agents can also be learning agents with the correct components. A learning agent requires a performance element, learning element, critic, and problem generator. The performance element selects what action to take given the world state. This can be done with any of the four previously described agent architectures. The critic rates the model's performance based on a performance metric. This metric is specific to the environment, such as distance to the goal or points earned. This is important because the world state does not provide an indication of the agent's success. The learning agent then makes improvements to the performance element based on the feedback from the critic. Lastly, the problem generator suggests new actions to the performance element that can lead to new information. Without this, the performance agent would always pick the best solution given its knowledge and would eventually stop learning. The problem generator allows for exploration, which might be suboptimal in the short run, to find better solutions for the long run.

While the learning utility-based agent has the highest flexibility in terms of agents, it is also the most complex to implement. For a given task it is important to properly examine the requirements before an agent model is selected. No single agent is optimal for all tasks. Where one task may require a goal-based agent, another may only require a simple reflex agent.

2.3 Test Environment

Space Navigator [5] is a tablet based route creation game designed for use in human-machine team experiments. In the game, the player is tasked with guiding colored spaceships to their corresponding home planet by drawing a trajectory for the ship to follow. The goal of the game is to get the most points possible within a five-minute gameplay period. An example of *Space Navigator* is shown in Figure 5.

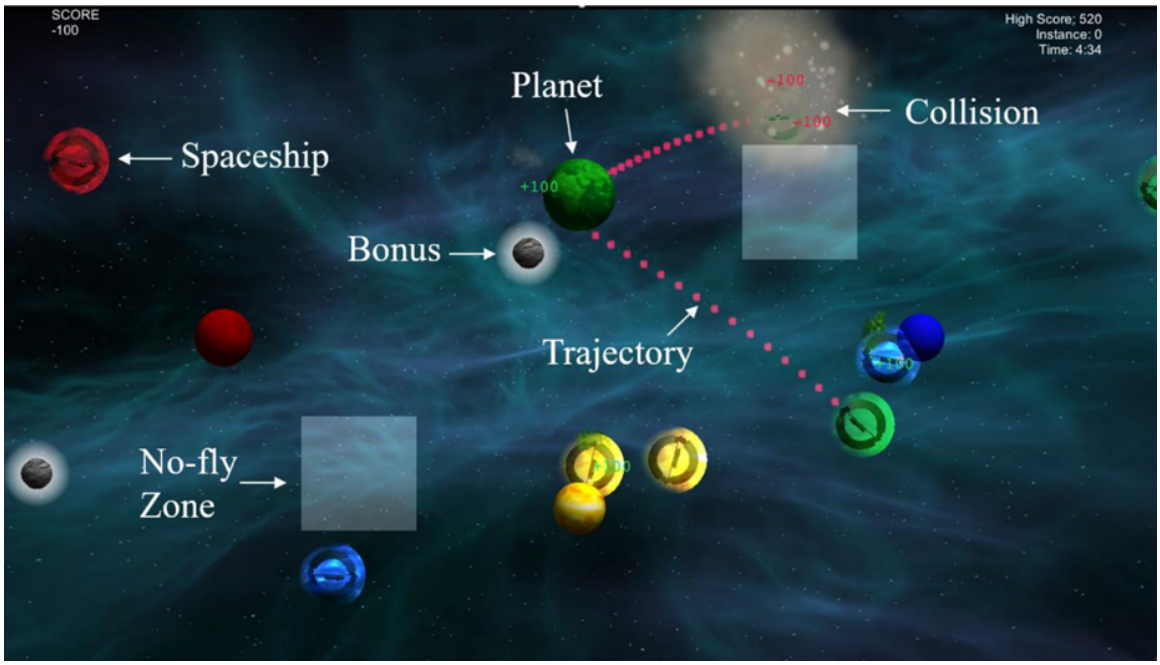

Figure 5. An example of a game of Space Navigator.

There are four primary objects in the environment.

- Spaceships: Ships spawn from the edge of the screen every two seconds. When spawned they have a random initial trajectory aiming them towards the opposite side of the map. The ships fly forward until a trajectory is drawn by the player. This is accomplished by selecting the ship with a finger and dragging the colored marker to create the intended trajectory. The ship will then follow the trajectory until it reaches the end. Ships are removed from the environment if they either: reach their home planet, collide with another ship, or fly off the screen. The side ships spawn on and color of ships is random.

- Planets: Planets are fixed objects on screen. Each of the four ship colors has its own corresponding colored home planet. When a ship touches its corresponding home planet the ship is removed from play. Ships can fly through planets belonging to other colors with no negative effects.

- Bonuses: Bonuses are small grey orbs that spawn randomly throughout the

map. A bonus will spawn at a random location on the map every ten seconds. Bonuses can be collected by ships to earn extra points. When a ship intersects with a bonus the bonus will be removed and points will be scored. Bonuses do not move or disappear after spawning until collected by a ship.

- No-Fly Zones: No-fly zones are stationary grey boxes that the player must avoid having ships fly through. There are two no-fly zones placed as specific locations on the map. A ship that enters the no-fly zone will lose a small number of points for each second it remains in the no-fly zone. Multiple ships can be inside a no-fly zone at once and each ship inside will lose points.

In previous research *Space Navigator* had four goals that can be defined under the taxonomy described earlier:

- Match(Ship, Planet): The player obtains 100 points when a ship reaches its corresponding colored home planet.
- Match(Ship, Bonus): 50 points are received when a ship matches its location with a bonus.
- Avoid(Ship, No-Fly Zone): 10 points are lost per second that a ship is inside of a no-fly zone.
- Avoid(Ship, Ship): When a ship collides with another ship, 100 points are lost per ship and both ships are removed from play.

2.3.1 Previous work using Space Navigator.

Previous studies have used Space Navigator for a wide array of experiments. *Space Navigator* has been used for studies of timing within human-machine teams [11], communication in adaptive environments [30], and reliance versus compliance with

automation [10]. None of the previous experiments utilizing *Space Navigator* have involved complete automation.

2.4 A* Search

An important function of the agents added to *Space Navigator* is the ability for the agents to play at and above human performance levels. Intelligent trajectory generation for the ships is a key part of accomplishing this goal. To that end the A* search algorithm is utilized for ship trajectory generation. More detail on the implementation of the algorithm is discussed in Chapter 3. Here the A* algorithm is reviewed.

A* is a best first search algorithm used for pathfinding or graph traversal [26]. Being a best first search, it is guaranteed to find the path with the smallest cost. The algorithm uses the function, $f(n) = g(n) + h(n)$, to calculate the cost of the path at each node. The variable n represents a specific node in the graph, $g(n)$ is the path cost from the starting node to node n, and $h(n)$ is the calculated heuristic estimating the cost of the path to the goal. So long as $h(n)$ is admissible (it never overestimates the cost to the goal node) A* finds the lowest cost path to the goal node. For grid searches $h(n)$ is often calculated using the Euclidian or Manhattan distance from n to the goal node.

The algorithm works by first calculating $f(n)$ for each of the nodes surrounding the starting node. New nodes are added to an open list and sorted by cost, where nodes with a low $f(n)$ are first. The first node on the open list, the node with the lowest cost, is popped and its surrounding nodes are evaluated. This continues until a path to the goal is found. Because each node stores its parent, the path from the goal to the start node can be recovered.

2.5 Concept Drift

In any supervised learning problem, the goal is to predict a target variable y given some set of input features X. The target variable y can be a continuous value in a regression task or discrete valued in a classification task. The learning problem is to use known (X, y) training pairs to teach the learner how to properly predict y for future inputs of X. In mathematical terms this problem can be written as attempting to find $p(y|X)$ [8].

For many learning problems it is assumed that the function $p(y|X)$ will not change with time. For example, a hypothetical image classifier that can perfectly identify human faces would not need to be updated because the look of a human face will not evolve significantly over any reasonable time scale. For many problems, however, it is not always the case that the data will remain stationary. A spam recognition algorithm, for example, needs to adapt to be able to recognize new sources of spam mail that did not exist when it was originally trained. In dynamic environments the data change over time, making the previously learned predictive model obsolete for the new distribution of data. The change in data over time is referred to as concept drift.

The key assumption associated with concept drift is that it is unpredictable to the learner [13]. The learner has no information about the source of the data thus it cannot predict the nature of the drift data. This is what differs concept drift problems from multitask learning, where all the data is known beforehand and an appropriate classifier can be trained for each concept. In concept drift the system has no ability to learn an appropriate predictor for the drift data before seeing it, thus the system must be able to adapt its predictions over time to the new data.

In this research, the goal changes made within the environment can be viewed as concept drift. An agent tracks the goal state and relates ship actions (X) to score

results (y). When the goal changes the system must be able to detect that the data has changed and learn the new (X, y) pairing. Concept drift literature presents a number of solutions for overcoming this detection problem.

2.5.1 Types of Concept Drift.

Concept drift can formally be defined as a change in probability between two times such that, $p_{t1}(X, y) \neq p_{t2}(X, y)$, where p denoted the joint probability of X and y at time t [8]. As described by Kelly *et al.* [17] concept drift can occur from several sources of change in the data:

1. The prior class probability, $p(y)$ may change.
2. The distribution of classes, $p(X|y)$ may change.
3. The posterior distribution of class membership, $p(y|X)$ may change.

These three types of drift are illustrated using a two class, two feature example in Figure 6. Figure 6_a shows population drift where the prior probability of a class, $p(y)$, has changed. This change lead to a class imbalance problem and additionally cause a classifier to become miscalibrated to the data [13]. Figure 6_b shows virtual concept drift. Here changes to $p(X|y)$ result from a change in $p(X)$. This change occurs without altering the decision boundary $p(y|X)$. Virtual drift can cause error in the learner from shifting the features into areas with no training examples [13]. Lastly real concept drift is illustrated in Figure 6_c. Real concept drift corresponds to a change in the posterior distribution of class membership $p(y|X)$, resulting in a change the decision boundary between the classes. This change can occur with or without a change to $p(X)$ [8].

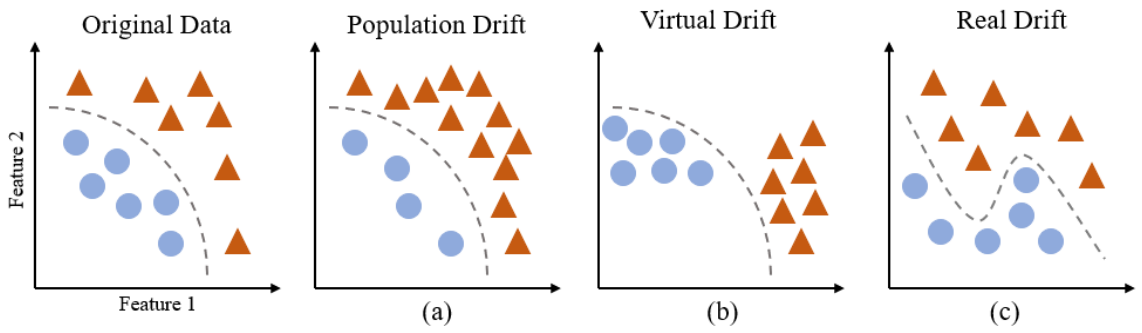

Figure 6. Example of the of the different drift types.

2.5.2 Patterns of concept drift.

In addition to the different types of concept drift, different environments also experience different patterns or speeds of concept drift. An important step in developing an adaptive learner to handle concept drift is identifying the pattern of drift. Each presents different challenges for an adaptive learner. The patterns are explained within the context of a system with two concepts C_1 and C_2. These patterns extend to any number of different concepts however.

The first and most basic pattern of concept drift is sudden drift. In sudden drift the target concept switches from C_1 to C_2 at some time. An example of sudden drift is a system that must adapt to new data from a sensor replacement [8].

Gradual drift is the second pattern of concept drift. Here there is a length of time when C_2 has an increasing probability of appearing in the data until completely becoming the target concept. Early in gradual drift, the instances of C_2 may be categorized as random noise as opposed to the start of drift [36]. An example of gradual drift is spam classification of emails from a website. When a user first signs up for promotional emails they are relevant, but as the user uses the site less the emails can more often be classified as spam.

A second form of gradual drift is incremental drift. Incremental drift differs from the previous pattern in that there are more than two levels for the target concept.

There are multiple levels of small differences between the targets. The concept slowly drifts between C_1 and C_2 along these levels so that the drift is only noticed when looking over a long period of time [36].

Lastly there are reoccurring concepts. This is when a target reappears after some time not being active. Reoccurring concepts are not always periodic and thus cannot be easily predicted in the same way basic seasonality could be [36]. For example, someone may have sudden interest in player statistics from a sports team, but only if the team won that week.

Figure 7 illustrates the different patterns of concept drift. Not all examples of concept drift are guaranteed to fall neatly into one of these patterns however. It is possible that the observed drift displays several of these patterns over time.

2.5.3 Taxonomy of an Adaptive Concept Drift Learner.

A system that can adapt to concept drift must be able to overcome two key problems [13, 8]. First, it must be capable of detecting the legitimate concept drift while filtering out noise in the data. After detecting, the system then needs the capabilities to adapt to the drift. To accomplish this goal the adaptive learner must accomplish three tasks. First, it must *predict* $\hat{y}_t$ from feature X_t using the stored predictive model L_t. Next, it *diagnoses* the prediction by calculating the loss after the true label y_t is received. Finally, the system can *update* its predictive model to L_{t+1} using the new sample (X_t, y_t). Figure 8 shows the generic framework for an adaptive learning system presented by Gama *et al.* [8] that accomplishes these goals. The four key components of the framework are the memory, learning, loss estimation, and change detection modules.

The four modules make up an adaptive learning system that can handle concept drift. The memory module handles how much memory is stored and when memory is

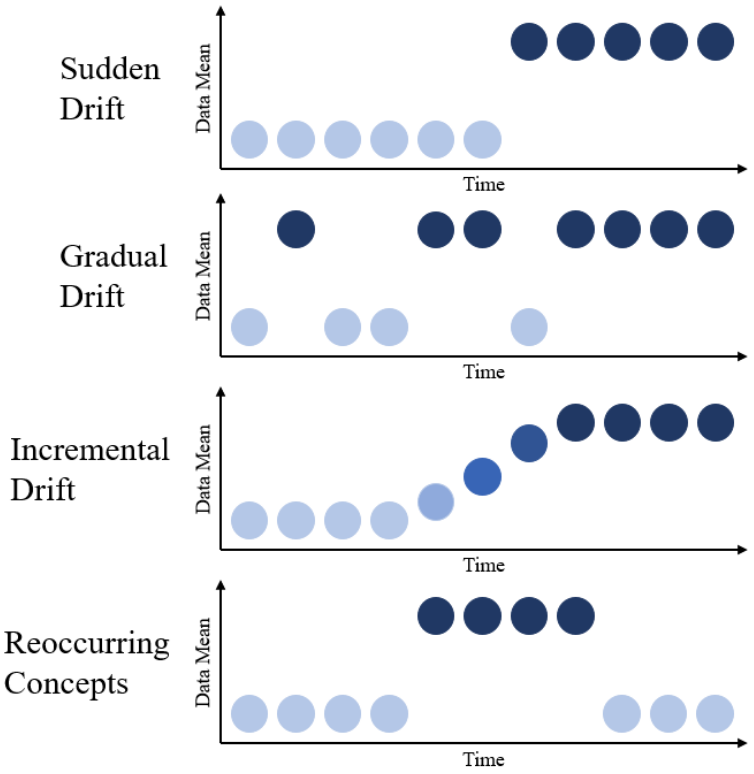

Figure 7. Example of the of the different drift patterns. Recreated from Zliobaite [36].

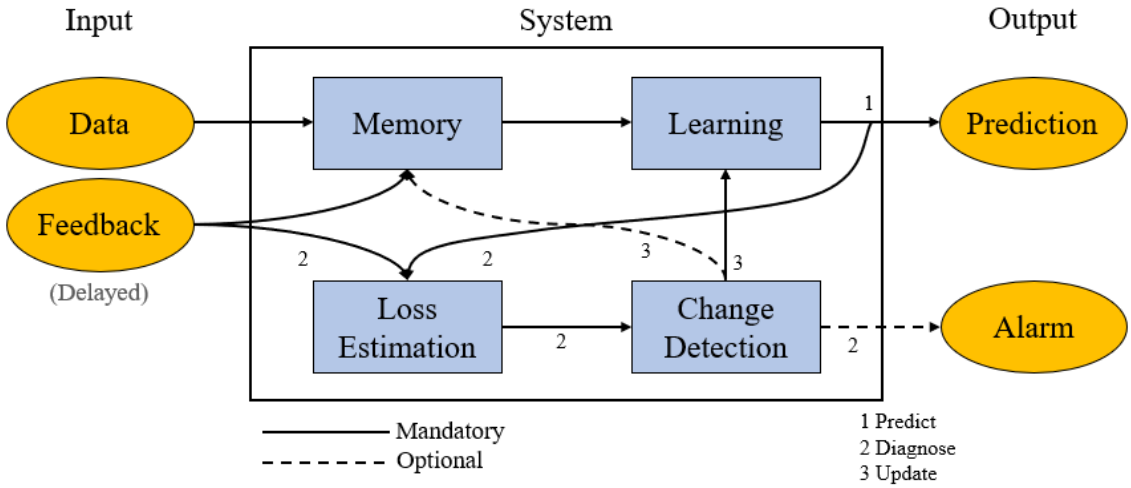

Figure 8. Generic framework for an adaptive learning system. Recreated from Gama *et al.* [8].

forgotten. Just as the model must constantly be updated with new information, old information must also be forgotten. The learning module handles the mechanisms for updating the model with new data. It controls how the model is updated, how the model reacts to drift detection, and if necessary, the handling of multiple ensemble models. Next, the loss estimation module provides feedback to the system about the quality of predictions. The system needs to account for possible delays between a prediction being made and when the true label is finally received. Lastly the change detection module is used to identify drift in the data using information from the loss estimation module. This will signal an alarm as to when drift has occurred. For each of these modules, design decisions are dependent on the problem domain and the data being analyzed. No one solution will work for all problem domains.

2.6 Reinforcement Learning

Reinforcement learning is the task of using observed rewards to learn the best action policy for the environment [26]. Reinforcement learning is used when the environment model and reward function are unknown. In the dynamic environment created for this thesis, reinforcement learning is utilized to train the agents on the goals of the environment. This section examines the components of reinforcement learning, exploration, exploitation, and how to switch between the two.

2.6.1 Exploration Versus Exploitation.

A principle component of reinforcement learning is the concept of exploration and exploitation. Exploration is the act of taking new actions in the environment to learn new information about the environment. Exploitation uses this learned knowledge to maximize the expected reward. The most common example of this dynamic is in the multi-armed bandit problem [29]. In this problem, there are a number of slot machines

which each provide a different payout based on some probability distribution. The payouts are unknown to a gambler, who is attempting to maximize the reward from a finite number of plays. The gambler must explore the environment by playing different machines to determine which one has the highest payout. Exploitation occurs when the gambler decides to use their exploration knowledge to only play the machine with the highest payout. In this problem, the gambler must balance exploration and exploitation to maximize the reward. If the gambler does not explore enough before exploiting, then their information could be incomplete leading them to exploit a lower paying machine. If exploration occurs for too long however then the gambler will lose rewards by continuing to play machines it knows do not provide the most reward. Finding a balance between the two is an important decision in reinforcement learning.

2.6.2 Methods of Exploration.

Exploration of the environment is important because the agent can only learn from what it has experienced. Broad exploration is required to ensure that the agent has enough knowledge of the environment to perform well. There are two methods of exploration: undirected and directed. Undirected methods use randomization to explore the environment [34]. Directed methods use knowledge of the learning process to explore in a guided manner.

Undirected exploration is used when there is no available knowledge of the learning process. This involves using randomized action selection to explore the environment. The most basic method is to assign each action a probability based on a uniform distribution then randomly select an action from the distribution. Alternatively, each action can be assigned a probability based on the actions expected reward, so that action's with a higher expected reward are explored more often. Undirected methods are useful if exploration costs are not factored into the exploitation phase.

Directed methods alternatively utilized memory of the learning process to have guided exploration. Two directed methods are error-based exploration and recency based exploration. In error-based exploration, actions that have had the largest change in their prediction error are explored more often. In recency-based exploration, a recency value is given to each object. The recency value tracks the length of time since the action has taken place. In the explore phase, actions with high recency value are prioritized for exploration.

2.6.3 Methods of Switching between Exploration and Exploitation.

Another important part of the exploration exploitation paradigm is knowing when to switch between the two phases. As mentioned earlier, it is important that a proper balance is maintained between the two phases to ensure the agent is maximizing its reward. Wilson [35] lists ten different techniques for switching between exploration and exploitation. The strategies involve calculating a probability that an exploration action will be taken. A selection of a few of the techniques are presented below.

The first method described is using a constant probability of exploration. That is, for each action taken there is a set probability that the action taken will follow the exploration rules as opposed to the exploitation rules. This method allows for precise control over how much time the agent explores. It does, however, suffer from a slow rate of exploration early in the system. Additionally, because the probability never changes, the agent will continue to explore even after it has learned everything about the environment.

Another strategy is to have a decaying probability so that the agent explores more at the beginning of the task and less as time moves on. This method solves the issue of continuing to explore after learning the environment. The down side however, is that after the probability decays the agent has no chance to explore again. The

approach works well in static environments where the agent will not need to reexplore the environment.

An adaptive strategy involves using a maximum system reward variable, R, to determine when to explore. R represents the maximum possible reward that can be obtained by the agent. This is compared against the reward performance of the agent to determine if there should be more exploration. The further the agent is from the maximum reward, the greater the chance of exploration. As the agent learns more, it will approach R and the probability of exploration will decrease. Unfortunately, this approach requires R to be known, which is not always possible for an environment.

The final of Wilson's approaches discussed here is the use of prediction error to determine when to explore. In this approach, the average prediction error for an action is tracked. Every time the agent takes an action the actual reward is compared against the predicted reward to calculate the prediction error. A high prediction error implies that the agent must explore more. As the prediction error decreases, so does the chance of exploration. This method allows the agent to decrease its exploration rate as its knowledge increases. This approach works in a dynamic environment because the prediction error will increase as the environment changes, thus increasing the agent's probability of exploration.

In an experiment of different exploration/exploitation control methods Rejeb *et al.* [24] propose a meta-rules-based approach to switching. In this approach, the agent explores for n periods and then exploits for m periods. Then if performance at time $t + n$ is greater than performance at time $t + n + m$ then the agent has more to learn so the number of exploitation periods is reduced by a fixed learning rate. If performance at time $t + n$ is less than performance at time $t + n + m$ then the agent has sufficiently explored the environment and the periods of exploration are extended by the learning rate factor.

2.7 Chapter Summary

This chapter presented an overview of how gameplay environments can be changed. The *Space Navigator* environment was discussed in detail. The topic of concept drift was presented, the various forms of drift were reviewed, and lastly how to handle concept drift. The chapter ended with a discussion of several different methods for exploring an environment and approaches to switching between exploration and exploitation. This background will be used in chapter III to develop a version of *Space Navigator* with dynamic goals and a collection of agents that will adapt to the dynamic environment.

III. Methodology

This chapter explains the methodology behind the updates to the *Space Navigator* environment and the experiment designed to test how different categories of goals affect an agents ability to detect and adapt to goal changes. The chapter begins with the changes made to *Space Navigator* to give it multiple goals. Next the method in which the goals are added and the exact changes each of those goals makes to the environment is discussed. Following that is an overview of the agents designed to play the adapted version of *Space Navigator*. Lastly, this chapter will cover the experiment designed to test and evaluate the performance effects of dynamic goals on agent performance.

3.1 Changes to Space Navigator

In order to examine how an agent performs in a dynamic goal environment, *Space Navigator* first had to be modified to have multiple goals. The four goals implemented into *Space Navigator* are the avoid, match, destroy, and create goals described by Djaouti *et al.* [1]. These goals were added to the game to provide a collection of meaningful changes that require a strategy adaption. A previous study with *Space Navigator* found that game dynamics changes did not require players to adapt their strategies [30]. The agents need to change their gameplay to adapt to the new goals designed for this research.

The new goal scenarios could not be added to *Space Navigator* as is. Due to the limited number of mechanics in the default version of *Space Navigator*, several changes had to be made to the environment for it to be able to support testing of changes between multiple goals. The ship object is too mechanically limited for meaningful new goals to be added. The changes made to the environment all work to give more

options for what the ship can do.

The first change implemented was giving each ship a shield. The shield allows ships to survive a single collision before being destroyed. When a ship has its first collision its shield is depleted. Then, on the subsequent collision, the ship is destroyed. This change allow for the implementation of more complex goals with further depth. Ships can now purposefully perform a collision, then continue to act afterward.

The next change reduced the number of spawning ship colors from four down to three. Green ships no longer spawn, and the green planet was removed from the map. This leaves only the possibility for red, blue, and yellow ships to spawn. The remainder of this thesis will refer to these three colors as the basic ships.

Coupled with the removal of the green ships, is the addition of a ship fusion mechanic. When two basic ships of different colors collide, a new ship with a fusion of the two colors is formed. A blue and red ship collide to form a purple ship, red and yellow form an orange ship, lastly blue and yellow form a green ship. When different colored ships collide, they are both removed from play and a fusion ship is spawned at the collision location. The fusion ship spawns with a random initial trajectory. Additionally, the fusion ships start with a shield just as the basic ships do. The fusion ships do not have a home planet under most goal conditions and can only be removed from play through collisions or by flying off the map. When a fusion ships collide with any other ship, both ships lose their shield or are destroyed if their shield has already been depleted. Like the first change, fusion ships expands the range of ship interactions possible in the environment. The fusion mechanic provides the opportunity for goals that require the player to create objects that do not naturally spawn within the environment. These changes provide *Space Navigator* with enough mechanical capacity to support the addition of new goals into the environment.

The last change made to the environment was removing the no-fly zones and bonus

pick-ups. Because neither of these objects are changed across the four goals, their removal allows for greater clarity about the decisions the goal agent is making. Left in, they would create noise in the final score, making it more difficult to determine the true performance of the goal agent. Additionally, previous research using *Space Navigator* has shown that players tend to ignore both objects, instead focusing solely on collision avoidance and trajectory generation [30].

3.2 Goal Design

Four new goals were added to the *Space Navigator* environment. The new goals added align with the four goal mechanics: avoid, match, destroy, and create, described in Chapter 2, Section 1. This section details the implementation of the new goals and relates them to the taxonomy diagram detailed in Chapter 2. The new goals do not add any new gameplay mechanics or change how the player will fundamentally play the game. Across the goals, the player will continue to draw trajectories for ships to guide them to parts of the screen. What the goals do alter, is the point reward for the component interactions. Switching between the goals will require the player to alter their play strategy to compensate for the change in how they will earn points.

To avoid any obfuscation on the effects on performance, goal changes were implemented only through modifications to the ship and planet component interactions. This means that between each goal only the points scored from ship/ship and ship/-planet collisions have changed. Table 1 summarizes the point breakdown for each interaction across the four goals. The points were distributed across the goals such that at most a single ship can earn 150 points and at worst lose 150 points. A description of the four goal scenarios (`avoid`, `match`, `destroy`, and `create`) are presented as follows:

Table 1. Breakdown of points earned by action for each goal.

Goal \ Action	Planet Collision	Shield Depletion	Ship Destruction	Ship Fusion (Per ship)
`Avoid`	150	-50	-100	-50
`Match`	50	100	-150	-75
`Destroy`	-150	50	100	-150
`Create`	100	-50	-100	100

3.2.1 Avoid.

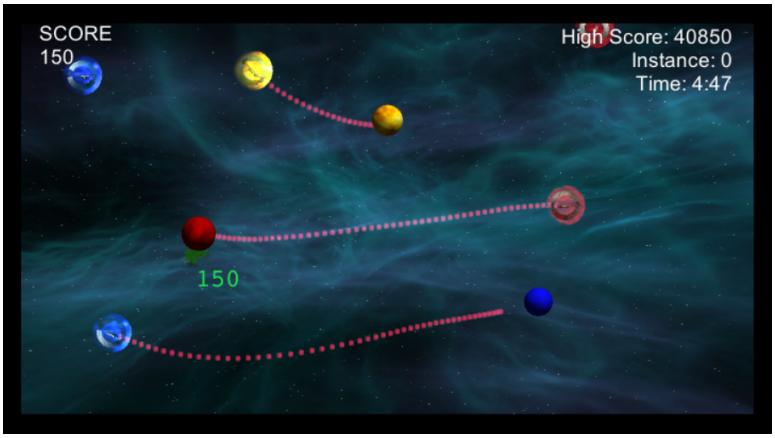

Figure 9. Example of the `avoid` goal scenario. The player earns points from landing the ship on itfls home planet.

The `avoid` goal is largely the same as the base goal in the default version of *Space Navigator* described in Chapter 2. This scenario is characterized by the need for ships to avoid collisions with other ships in order to maximize score. The player losses points from all forms of ship collisions: -50 points when a ships shield is depleted, -100 points when a ship is destroyed, -50 points per ship when a fusion ship is created, and 150 points are gained when a ship lands on its home planet. An example of the `avoid` goal is shown in Fig 9. In relation to the change taxonomy the `avoid` goal is a goal mechanic with a depth and breadth of one. The player only needs to guide the ships to their home planets.

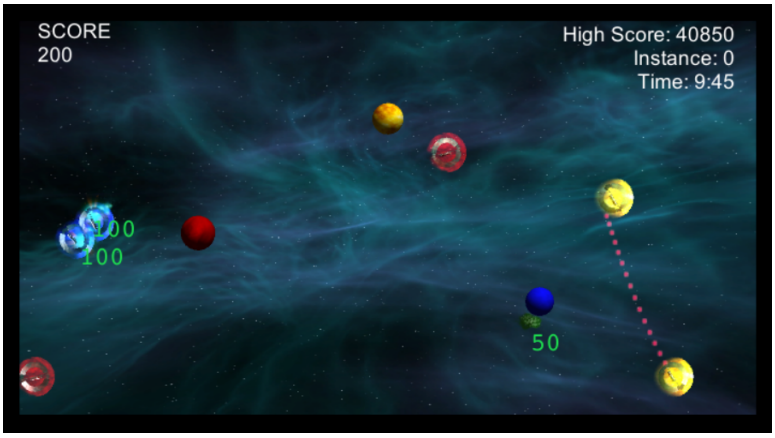

Figure 10. Example of the `match` goal scenario. The player earns points by colliding the two blue for the first time and landing a ship on itfls home planet.

3.2.2 Match.

The `match` goal tasks the player to match ships of the same color to deplete the ship's shield. In this scenario the player earns: 100 points when a ship's shields are depleted, 50 points if the ship lands on its home planet, -150 points if a ship is destroyed, and -75 points per ship from a ship fusion. An example of the `match` goal scenario is shown in Fig 10. The `match` goal is goal mechanic with a depth and breath of two. To maximize their score a player needs to first deplete a ships shield, then guide it to it's home planet. The `match` goal has added breadth, however, because the player can land ships on planets without depleting it's shield first.

3.2.3 Destroy.

In the `destroy` goal, the player earns points from ships destroyed through multiple collisions. The player gains: 50 points are gained when a ships shields are depleted, 100 points when a ship is completely removed from the map, 150 points when a ship lands on its home planet, and -150 points per ship when a ship fusion occurs. An example of the `destroy` goal scenario is shown in Fig 11. The `destroy` scenario is a breadth one and depth two goal mechanic. The player must first deplete a ships

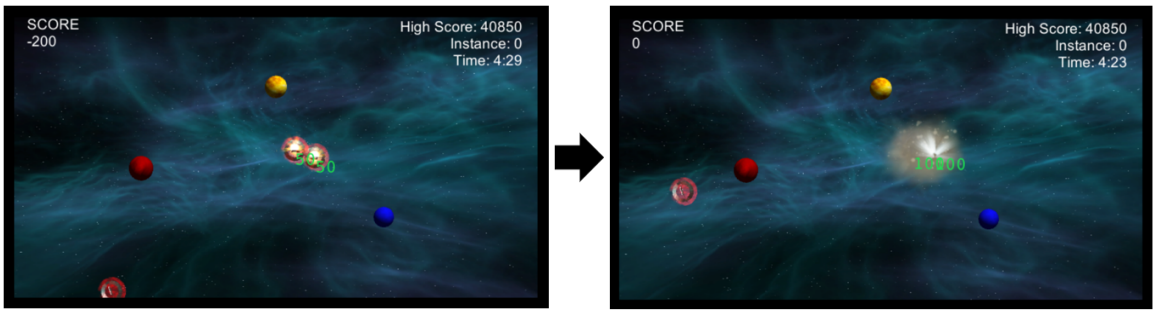

Figure 11. Example of the `destroy` goal scenario. The player earns points from first, depleting both red shipsfl shields and then for both ships being destroyed from another collision.

shield before it can be destroyed.

3.2.4 Create.

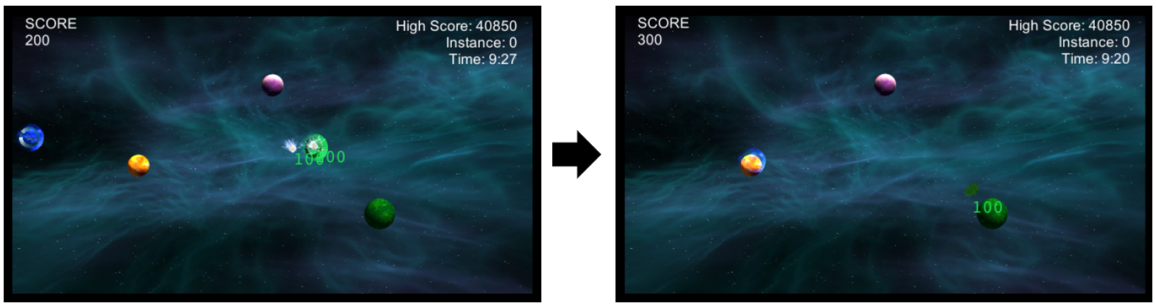

Figure 12. Example of the `create` goal scenario. First a green ship is formed through the collision of a blue and yellow ship. Then points the player earns points from landing the ship on the green planet.

In the `create` goal scenario, the home planets for the basic ships are removed from the map and the fusion color home planets are put in their place. The goal then is to match ships of different colors to form fusion ships and then land those fusion ships on their corresponding home planets. The player gains: 100 points per ship from a ship fusion, 100 points if a ship lands on its home planet, -50 points when a ships shields are depleted, and -100 points when a ship is destroyed. An example of the `create` goal scenario is shown in Fig 12. Like the `destroy` goal, `create` is also a breadth one, depth two, goal mechanic. The player must create a fusion ship in order to land ships on planets.

3.2.5 Goal Implementation.

Across the four new goals, two are actually achieved through repetition of the match goal mechanic. The `destroy` and `create` goals are accomplished through avoid and match mechanics. Due to the gameplay mechanics of *Space Navigator* the true goal mechanics of destroy and create could not be added. The destroy goal mechanic generally requires an object not controlled by the player that the player is then trying to destroy. The create goal requires a resource that the player spends to create new objects. In real time strategy games for example gold is accrued by the player and then spent to create units. The idea of having the player shoot asteroids for the destroy goal and spending points to spawn a fusion ship for the create goal were initially considered, but the ideas were ultimately thought too complicated for the player to balance with the other tasks required within the environment. The destroy and create goals have thus been abstracted through the other two goals. The `destroy` goal is a match followed by another match. The `create` goal is accomplished through a match then an avoid. While the `match` and `create` scenarios have the same subgoals for optimal points, a key difference between the two is that in the `create` goal the player does not have the option to land ships on a planet prior to the completion of the match subgoal. This helps make each goal distinct, even though they share the same goal mechanics.

3.3 Agent Design

The "player" for this experiment is a collection of three agents coded to play *Space Navigator* across all of the goals and adapt when changes occur. A line agent and avoidance agent work in conjunction to play *Space Navigator* at a high level with few errors. Meanwhile, a goal agent agent designed to track the goal of the environment, detect when the goal changes, and assigns targets to aforementioned

line and avoidance agents. The line and avoidance agent provide autonomous play of *Space Navigator*. The goal agent will be used to answer the research question of whether different categories of goals affect the ability to detect and adapt to goal changes.

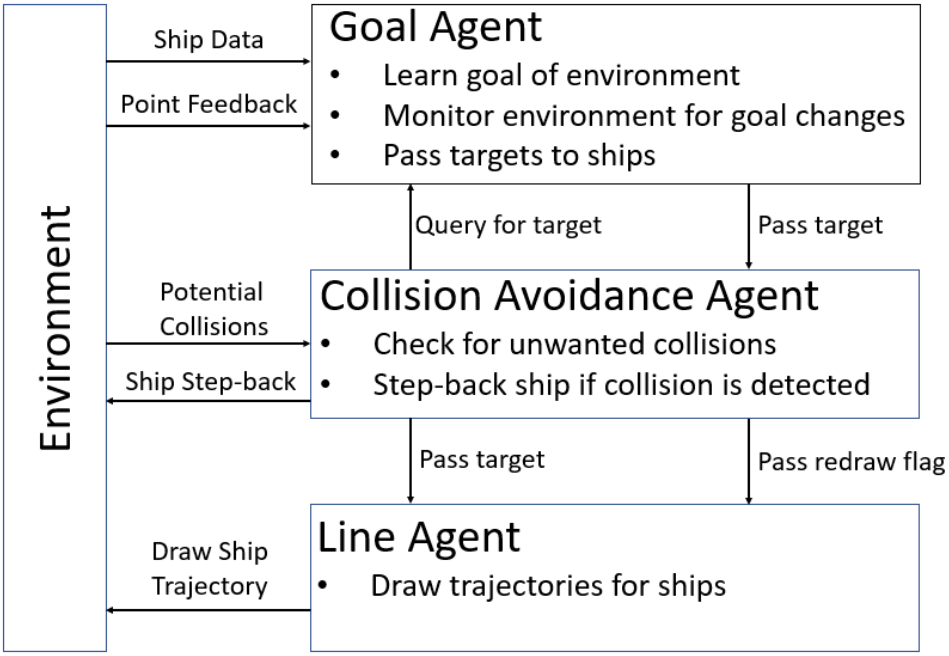

Figure 13. Interaction diagram for the three *Space Navigator* agents.

3.3.1 Line Agent.

The line agent outputs a trajectory for the ship to follow. To be successful, the agent needs to quickly determine a path to the intended target and update the path to compensate for changes in the environment. Two different line agents were created and tested for this purpose.

The first line agent was a simple straight-line agent. The agent would accept a target object and then draw a straight trajectory directly to the object. The straight-line trajectory was extremely efficient to create and always provided ships with the fastest route to their target. In practice, however, this agent suffered from multiple

short-comings. Most importantly, the agent had no way to path around the other planets and ships. The straight line agent relied on a collision avoidance agent to move the ship around another object. The effect of this behavior was that the agent would most often create trajectories through the center of the map. Then, because there was so much traffic through the center, ships clustered up and the collision avoidance agent was not able to properly route the ships around the traffic. This behavior showed the necessity for more intelligent trajectory generation from the line agent.

The new line agent had to be able to create trajectories that avoided congested areas of the map and reduce the reliance on the collision avoidance agent. The A* pathfinding algorithm was chosen for a variety of reasons. A* has been shown to work well for pathfinding and is widely used across many video games. For example the popular games, *StarCraft* [7] and *Dragon Age: Origins* [4], both use variations of A* for unit pathfinding [2]. It was additionally choosen due to how easily it could be added to *Space Navigator* through outside asset packages.

Instead of writing an A* pathing agent from scratch, the free A* Pathfinding Project Unity package by Aaron Granberg [12]. The A* Pathfinding Project pro version is the top-rated pathfinding solution in the Unity assets store and has been used in several commercial games [12]. The free version of the package only removes several specialized features that are not required for this project.

Seen in Figure 14, the *Space Navigator* map has a 32x64 grid placed over it. This is the grid over which the A* algorithm performs the search. The grid dimensions were chosen to balance resolution and performance. Larger grids with more nodes take too many resources to update while smaller grids did not provide enough resolution for the agent to make good trajectories.

To have the A* algorithm avoid congested areas of the map, each game object has

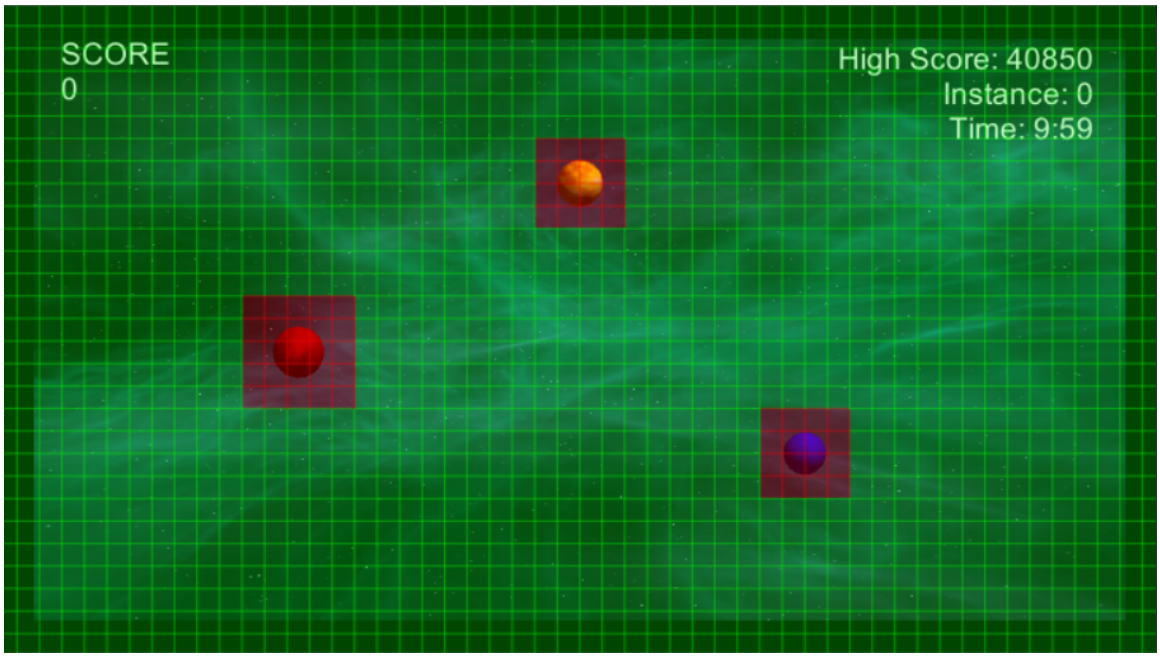

Figure 14. Grid used for A* pathfinding. The red cells around the planets show penalty zones.

a zone of influence. These zones add a penalty to all overlapping nodes, raising the path cost for future A* agents that are searching the grid. This causes the agents to avoid these areas when pathing a route to the target object. Thus, when the A* algorithm is searching for a route to the target it avoids making a path directly along the path of another ship. As seen in Figure 14, each planet has a penalty zone around the planet causes the agents to avoid the planet. If a planet is the the target of a ship, then the A* pathing will accept the penalty and draw a trajectory to the planet, but will otherwise avoid drawing trajectories near planets. That way planets will be clear for ships that are trying to move there. Ultimately, the penalty zones define where ships will be and make it clear to other ships that those areas of the map should be avoided. Figure 15 shows an example of the pathing behavior of the A* agent. As seen, the agent avoids drawing trajectories close to other trajectories, allowing ships to move around each other with minimal chance of collision.

Each ship has its own line agent attached to it. There is not a singular agent at

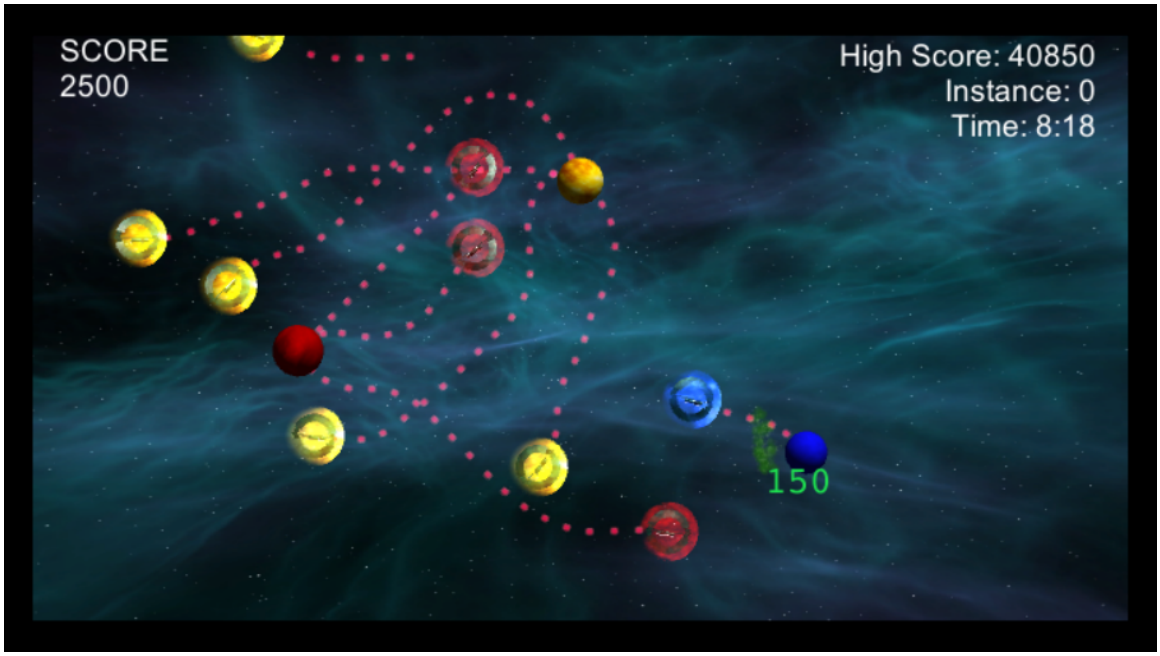

Figure 15. Example of trajectory generation with the A* line agent.

the top level of the environment drawing trajectories for each ship. Whenever a ship spawns, a copy of the A* line agent is also spawned and attached to the ship object. While the line agent has not target the A* agent simply stays at the ships location. After a target has been passed to the ship, the agent will then find a path to the target and begin to move along the path. Path markers are regularly instantiated as it travels along its route. When the A* agent reaches its destination, it is deleted. When a ship has another ship as the target, the agent paths to the target ship's line agent, and they meet in the middle. When a ship avoids a collision, a new line agent is spawned and the ship's path is redrawn. This allows ships to update their path around obstacles that were not there when the path was originally created.

3.3.2 Avoidance Agent.

The avoidance agent provides ships with the capability to avoid other objects in the environment. The pseudocode for this algorithm is shown in Algorithm 1. A version of this agent was originally built in [10], which used a step-back behavior to

avoid ship/ship collisions. Whenever a ship collision was about to occur, the ship would move back several steps then continue forward again. This, however, is an imperfect solution to the problem required for this thesis because it only accounts for ship/ship collisions. Additionally, when the ship was stepping back to avoid a collision, it would not simultaneously check to see if the stepping back location was clear of other ships. This resulted in ships causing collisions by stepping back into other ships in an attempt to avoid a collision.

Algorithm 1 Avoidance algorithm for avoiding unwanted collisions

```
1: Start:
2: target = null
3: goalAgent = Goal Agent Pointer
4: lineAgent = Ship Line Agent Pointer

5: function CheckIfTarget(raycastHit)
6:     if raycastHit == target then
7:         Return(true)
8:     else
9:         Return(false)
10:    end if
11: end function

12: function StepBack
13:    moveLocation = current position + (0,0,-1)
14:    MoveTo(moveLocation)
15:    lineAgent.resetTrajectory()
16: end function

17: Update:
18: if target == null then
19:    goalAgent.GetTarget(self)
20:    lineAgent.PassTarget(target)
21: end if
22: raycastHit = Spherecast object collision
23: if raycastHit != null && !CheckIfTarget(raycastHit) then
24:    StepBack()
25: end if
```

The implemented avoidance agent can avoid all objects in the environment. The

agent starts by querying the goal agent for a target object. The target is then passed from the collision avoidance agent to the line agent. Having one agent in charge of getting the target from the goal agent ensures that both the collision avoidance and line agent always have the same target. For the collision avoidance agent, the target is the one object that the collision avoidance agent will not avoid, thus allowing the target to be reached. The agent works to avoid collisions with all other target objects. If the agent is not given any destination object then it will simply avoid all objects in the environment until the ship eventually flies off the map, at which point it is removed from the game.

Spherecasts are used for collision detection. Spherecasts work by projecting a sphere along a ray and returning the first object that the sphere collides with [33]. They are a computationally efficient approach to detect future collisions. On each update cycle, Algorithm 1 line 22, a sphere cast (the size of the ship) is projected a full ship's-length in front of the ship. If the sphere cast collides with an object, the hit object is returned. This lets the ship know if it is going to collide with anything directly in front of it. The hit object is then compared against the target object to determine if they are the same. If they are, then the collision is allowed to occur, otherwise avoidance occurs.

Two behaviors were created for collision avoidance. The first was utilized in conjunction with the straight-line version of the line agent. In this version, avoidance was accomplished based on the Y position of the two objects. If the ship has a lower Y position than the object it is going to collide with, then the ship moves down. If it has the higher Y position then it moves up. Up and down in this context refers to movement along the Y axis relative to the ships rotation, not the Y axis as defined by the map. This movement behavior allows ships to navigate around both moving and stationary objects. While this behavior worked well for single pairs of ship's,

when, more than two ships were in the same area, the behavior did not help ships path around one another. When this behavior was tested with the A* line agent it would often push ships into the path of other ships. Thus, it was found that a simpler behavior improved performance. By reverting to a simple step-back agent, a ship creates distance between itself and the potential collision and then relies on the A* pathing to guide the ship around the object due to the penalty zones.

To improve on the previous step-back agent from [10], ships continue to actively avoid collision even while already avoiding a collision. Instead of blindly stepping back the ship first turns and checks if the location it would be moving to is open. If the ship would have a collision with another ship whilst already avoiding a collision, it will also avoid the new collision. This behavior causes the ship to stay in place by quickly stepping back between the two potential collisions. Finally, when a ship avoids a collision it sends a flag back to the line agent that it should redraw the trajectory. This allows ships to avoid objects that may have traveled into their path.

3.3.3 Agent Performance.

Tests of the line and collision avoidance agents were performed to determine baseline performance across the four goals. The two agents were provided the correct goals at the start of the game and played for five minutes. Thirty games were run for each goal. Figure 16 shows the results of the baseline trials. In a five-minute game, the maximum possible score is around 21,750, dependent on the final ship spawn locations. The figure shows that the agents perform best in the `avoid` and `match` goals, scoring close to the maximum at an average score of 20,875 and 20,300 respectively. The agents score roughly 2,000 less points when tested on the `avoid` and `destroy` goals. This is likely due to the fact that ships are on screen longer in these goals, resulting in more ships on screen and more unintended collisions.

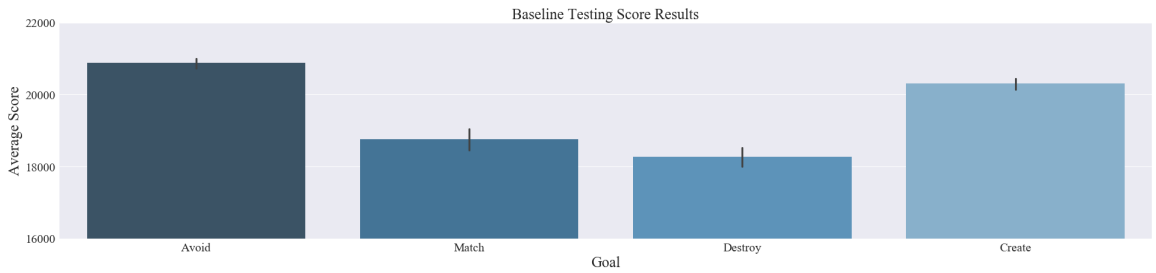

Figure 16. Score results from agent baseline performance tests.

3.3.4 Goal Agent.

The goal agent is the central agent that tracks the goals, assigns targets to the line and avoidance agents, and adapts to changes in the environment. The goal arbiter utilizes the adaptive learner framework presented in Section 2.6 to detect concept drift in the goals. Additionally, the agent utilizes exploration to search the environment and create a predictive model. Exploitation then utilizes this model to maximize score. The goal arbiter begins with no knowledge of the environment and is reset between all game instances. Psudocode for the goal agent is shown in Algorithm 2.

3.3.5 Adaptive Learner Framework Implementation.

As discussed in Section 2.6 and seen in Fig 17, for an adaptive system to compensate for concept drift it requires distinct memory, learning, loss estimation, and change detection modules. Before these are designed, the type of drift present in the environment must be analyzed. In this environment, prediction target Y is how many points will be earned from a specific ship interaction, X. Here the features of X are the ship color, shield level, target object, and target shield if applicable. The drift between goals is sudden drift. At the change point, the goal switches from one to another instantly and then does not change again for the rest of the game.

The memory module is implemented by using single instance memory. Only the most recent sample of each interaction is stored in memory. Because the data is

Algorithm 2 Adaptive Learner Agent algorithm for detecting change and assigning targets.

Start:
$exploreTime = currentTime + 45$
$Exploring = true$

function GETTARGET($ship$)
    if $Exploring = true$ **then**
        target = unexplored interaction object
    else
        Compile list of targets
        if Target exists **then**
            target = object
        else if Target not assigned in 20 sec **then**
            target = random object on screen
        end if
    end if
    return(target)
end function

function CHECKDRIFT($feedback, prediction$)
    Update memory
    Calculate loss
    Update loss windows
    if $avg(5\ sample\ Loss) > avg(10\ sample\ Loss) + 25$ **then**
        Clear loss windows.
        $exploreTime = currentTime + 45$.
        $Exploring = true$
    end if
end function

Update:
After Delay: $checkDrift(feedback, prediction)$
if $Exploring$ && $currentTime > exploreTime$ **then**
    $Exploring = false$
    Compile exploration knowledge
    Formulate goal
    Distribute goal across all ships
end if

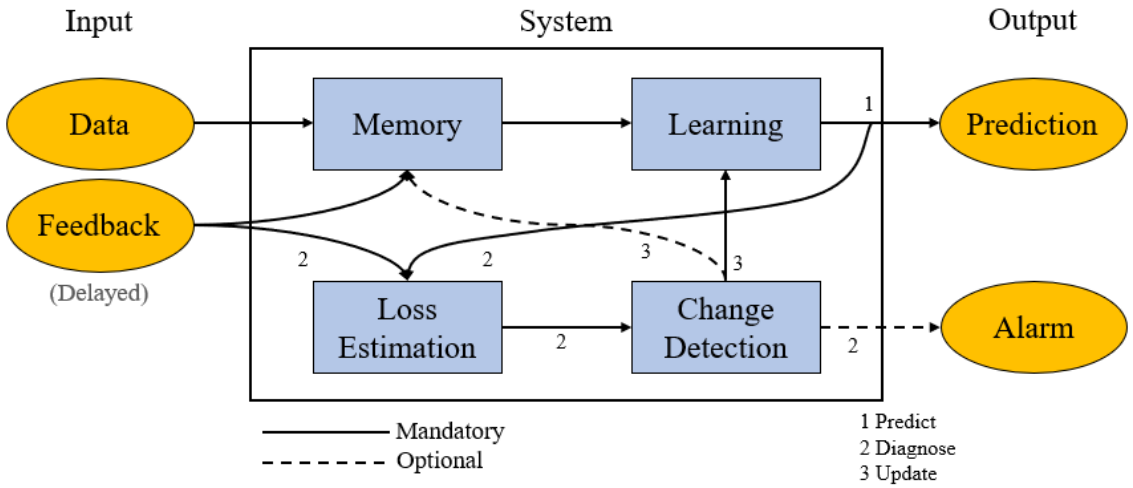

Figure 17. Adaptive learner framework used to create goal agent. Recreated from Gama et al [8].

noiseless, the score can be predicted from a single instance sample of an interaction. The most recent samples are more accurate than older samples and thus continually overwrite the previous sample for each X.

The learning module uses a predictive model generated by forming a look-up table between X and the true y feedback from memory. When a ship has a target assigned, the lookup table is used to determine how many points were earned from the last instance of the interaction and this value is used as the predicted number of points for the interaction. The learner uses local replacement to update the predictive model. As new samples are collected, they overwrite the old samples which are discarded from memory.

When a ship completes its assigned interaction, the agent receives a noisy value for the number of points earned. The noise added to the score is drawn from a unit normal distribution then multiplied by 25 to increase its effect. The noise avoids giving perfect information to the agent that would allow it to instantly detect when a change occurs. This represents how a player may not notice the change due to their attention being taken up from drawing trajectories and avoiding collisions. The noisy

score feedback is compared against the predicted number of points for the interaction to get the prediction error. The agent maintains a short (five sample) and long (10 sample) window of the error to compare the loss over time. The average loss in the short window will be affected more quickly after a change than the long window. When the average error in the short window is 25 points greater than the average loss in the long window a change alarm is triggered. The alarm signals to the agent that it should begin exploring the environment to learn the full extent of the changes.

3.3.6 Exploitation.

In addition to the adaptive learner, the goal agent utilizes exploration and exploitation to determine when and how the environment should be explored versus when the learned knowledge should be utilized for points. The goal arbiter can be in either exploration or exploitation mode depending on feedback from the adaptive learner.

During exploitation the goal arbiter utilizes the predictive model generated by the learner to assign targets to the ships that spawn. Whenever a ship spawns, the line agent and the avoidance agent query the goal arbiter for a target object. The goal arbiter uses the lookup table to determine what interaction will give the most points. If the interaction requires an object that is not currently present in the environment, then the goal arbiter will wait to pass a target object until the correct object has spawned. When an object that satisfies the interaction spawns or becomes available, the goal arbiter passes the object to the agents. If a ship reaches its first target and has not been destroyed, then it re-queries the goal arbiter for another destination object.

3.3.7 Exploration.

The goal arbiter uses exploration to inform the predictive model. It works to assign a variety of interactions to gain an understanding of how the points are distributed across the environment. Exploration is accomplished through a combined, random and recency approach. Randomness is introduced in exploration through the ship spawn system. The color of the spawning ship is a random uniform distribution between the three basic colors. This means that there is no way to determine when ships of a certain color will spawn. Thus, the exploration is bounded by the randomness of the ship spawns. If a blue ship does not spawn during the exploration period then there is no way for the goal arbiter to learn the reward structure for blue ships.

Recency is used in exploration to determine if the goal arbiter should explore an action or not. Whenever an interaction occurs, the time stamp is also recorded. When the goal arbiter is exploring the environment it first checks to see if the interaction has already been explored. If it has, then it checks the time stamp to determine how long ago the interaction was explored. Only interactions older than 30 seconds are explored again. The more recent interactions are ignored as the predictions are assumed to be correct due to how recently the data was collected.

Thus, when a ship spawns during an exploration phase the goal arbiter searches for the first available object that will result in a new point reward prediction occurring. However due to the randomness of the spawn there is no way to determine the order these will occur or which interactions will take place. For this reason, at the end of each exploration phase the predicted rewards are shared across all ships with missing interactions. For example, if by the end of the exploration phase there was no record of the reward from a blue ship interacting with a bonus, but there was one for yellow and a bonus, then the prediction for the yellow/bonus interaction will be shared to the blue ship. This assumes that different colored ships are likely to have the same

goals. If two ships have entries for the same interaction with different point value than the pessimistic approach is taken and the lower point reward is shared.

3.3.8 Switching Methods.

Switching between exploration and exploitation is dictated by time and the adaptive learner. While in exploration mode the goal arbiter will explore the environment for a fixed 45 seconds. Then the goal arbiter will switch to exploitation mode. While in exploitation mode the adaptive agent can trigger exploration again when drift is detected. This starts another 45 seconds of exploration.

It should be noted that usually exploration should be continuous, even if infrequent, in most systems. This is because drift could occur outside actions being exploited. Without exploration the system may never notice the drift and continue exploiting potentially suboptimal actions. Across the four goal the points change such that the positive point action always changes between goals. This ensures that the goal agent will always detect the drift after a goal change.

3.4 Experimental Design

3.4.1 Objective.

In the proposed experiment, the agents play *Space Navigator*, then at some time mid-game the goal switches to one of the other three goals. The objective is to determine the effects of goal changes on agent performance. The experiment determines if any one of the four goals is more difficult to learn and adapt to. The hypothesis for this experiment is that there will be a difference in the performance based on the type of goals, thus the null hypothesis is that goal switching has no effect on the outcome of the game.

3.4.2 Response Variables.

The response variables being collected are summarized in Table 2. Average Regret and average ship point rate will both be calculated after the experiment by processing the raw data output from the game of *Space Navigator*. There will not be any precision error in the measurements due to the experiment being fully simulated.

Table 2. Response Variable Summary

Response variable	**Normal operating level and range**	**Measurement precision**	**Relationship of response variable to objective**
Total Score	[-45,000, 45,000] Points	Exact measurement	Direct measure of performance for the trial
Detection Time	(0, 300] Seconds	Exact measurement to 0.01 seconds	Measure of goal agent perfromance
Learn Time	(0, 45] Seconds	Exact measurement to 0.01 seconds	Time taken until goal agent has enough information to determine the new goal

Total Score.

This is a measure of the total number of points scored at the end of the 10-minute game. The point distributions have been standardized so that the total number of points available is the same across all goals. In each goal condition the ships can earn a maximum of 150 points. Total score provides a simple measure to quickly understand the performance of the agents for a given set of conditions.

Detection Time.

Detection time is the time it takes the goal agent to detect a goal change after it has occurred. This is the primary measure of performance for the goal agent. The faster the detection time, the better the goal agent is performing.

Learn Time.

Learn time is the length of time, after a change has been detected, that it takes the goal agent to learn what the new primary goal is. The goal agent explores for a fixed length of time after a change is detected, however this measure records how soon into the 45 second period the agent has sufficient information to determine the goal. Because all ships share the same goal, once the goal arbiter has tested each action it will have learned enough to determine the goal. Analysis of the difference in lengths of time between different goal changes will reveal which changes are more difficult for the agent to adapt to.

3.4.3 Control Variables.

Although there are many variables in *Space Navigator*, this experiment is primarily concerned with the effects of goal changes and the length of time before the goals change. Table 3 shows a summary of the control variables and the proposed changes.

Table 3. Control Variable Summary

control variable (units)	**normal level & range**	**proposed settings, based on predicted effects**	**predicted effects (for various responses)**
Goal - A	-	Match, Destroy, Create	Difference
Goal - B	-	Avoid, Match, Destroy, Create	Difference
Change Time	0	150, 300	Unknown

Goals.

Each combination of goal pair changes will be tested, resulting in 12 unique goal change scenarios. In each trial, the agent will start with no knowledge of the goals. Goal-A is that starting goal and Goal-B is the goal that is switched to after the change

time is reached. This experiment is not concerned with more than two goal changes per trial because multiple goal changes can be seen as linking 2 trials together.

Time to Change.

This is the length of time, in seconds, that passes until the goal switches from goal-A to goal-B. There are 2 different change times being tested, 300 seconds and 150 seconds. The 300 second switch time will change the goal at the half way point in the game. It will provide feedback on performance when both goals are given equal time in the round. The 150 second trial will test the system's agents' ability to adapt to a quick goal change. Additionally, after the change it tests the performance of the agent when the goal remains steady for a longer period.

3.4.4 Held Constant Factors.

The held-constant factors, shown in Table 4 within the environment are the variables that remain constant across all trials. The ship spawn rate and game length are constant across all trails to ensure that the same number of points are available,.

Table 4. Held Constant Factors Summary

Factor (units)	**Desired experimental level & allowable range**	**How to control (in experiment)**	**Anticipated effects**
Ship spawn rate	2 ships every 2 seconds	Set in environment	Total points available in the environment
Game Length	600 Seconds	Set in environment	Total points available in the environment

3.4.5 Nuisance Factors.

Nuisance factors are limited due to the experiment being fully simulated. Most all of the factors can be accounted for by hard coding them to a set value. The one

nuisance factor in the experiment is the random ship spawn distribution. Because the ship spawning in *Space Navigator* is completely random, some scenarios may have more possible points than other due to how the ship spawns. For example, if a ship spawns on the opposite side of the screen from its home planet, there may not be enough time left in the game for the ship to reach its planet, thus those points could not possibly score. This causes variability in total points available in the environment. This variance should be minimized by running a sufficient number of trials of each condition.

Table 5. Nuisance Factors Summary

Nuisance factor (units)	**Strategy (e.g., randomization, blocking, etc.)**	**Anticipated effects**
Ship spawn distribution	Repeated trials	Variation in total points

3.4.6 Design Matrix.

From the two control variables 24 different test conditions have been tested. Table 6 shows the testing schedule for the 24 conditions. Each condition was run 30 times to ensure that enough data has been collected for statistically significant results.

3.4.7 Apparatus.

To avoid any issues with computational limitations, the trials were run in real time. With each trial being 10 minutes, 24 different conditions, and 30 captures for each condition, running the experiment on a single computer takes 120 hours in total. To speed up the data capture trials was spread across six different computers. This sped up the time taken to capture the data down to only 20 hours.

Table 6. Testing Matrix

Condition	Goal-A	Goal-B	Change Time	Total Score	Detection Time	Learn Time
1	Avoid	Match	300	-	-	-
2	Avoid	Destroy	300	-	-	-
3	Avoid	Create	300	-	-	-
4	Match	Avoid	300	-	-	-
5	Match	Destroy	300	-	-	-
6	Match	Create	300	-	-	-
7	Destroy	Avoid	300	-	-	-
8	Destroy	Match	300	-	-	-
9	Destroy	Create	300	-	-	-
10	Create	Avoid	300	-	-	-
11	Create	Match	300	-	-	-
12	Create	Destroy	300	-	-	-
13	Avoid	Match	150	-	-	-
14	Avoid	Destroy	150	-	-	-
15	Avoid	Create	150	-	-	-
16	Match	Avoid	150	-	-	-
17	Match	Destroy	150	-	-	-
18	Match	Create	150	-	-	-
19	Destroy	Avoid	150	-	-	-
20	Destroy	Match	150	-	-	-
21	Destroy	Create	150	-	-	-
22	Create	Avoid	150	-	-	-
23	Create	Match	150	-	-	-
24	Create	Destroy	150	-	-	-

3.5 Chapter Summary

In summary, this chapter covers the methodology for the changes made to *Space Navigator*, the design of the agents, and the experimental design. The mechanical capacity of *Space Navigator* was increased through the addition of more ship interaction mechanics. This allowed for the implementation of the four goals into the environment. Three agents were designed to both play *Space Navigator* and adapt to the changing goals. Concept drift methods and reinforcement learning were utilized to learn the goals and adapt upon changes. Lastly, the chapter presents an experiment

to analyze the effect that dynamic goals has on agent performance.

IV. Experiment One

This chapter presents the results of the experiment proposed in Chapter 3. Results focus on goal agent performance and how different goals affect overall system performance. Section 1 analysis the score results of the experiment. Section 2 details the change detection performance. Section 3 further analyzes the detection time results by looking at the relationship between detection time and goal similarity. These results are used to answer the thesis question showing how different categories of goals affect detection and performance.

4.1 Score Results

The score results present a general overview of the performance across the 24 conditions. Figure 18 shows the scores for the 12 goal combinations at the two change times. Across the 720 games, the maximum score was 36,100 points (occurring once in the `avoid-create` and `destroy-create` conditions) and the minimum score was -1,200 points (in the `match-avoid` condition). The average score over all the conditions was 30,052 points. The `create-avoid` conditions with a quarter change time achieved the highest average score of 32,990 points. The lowest average score of 26,825 point was obtained from the `destroy-match` with a half change time condition.

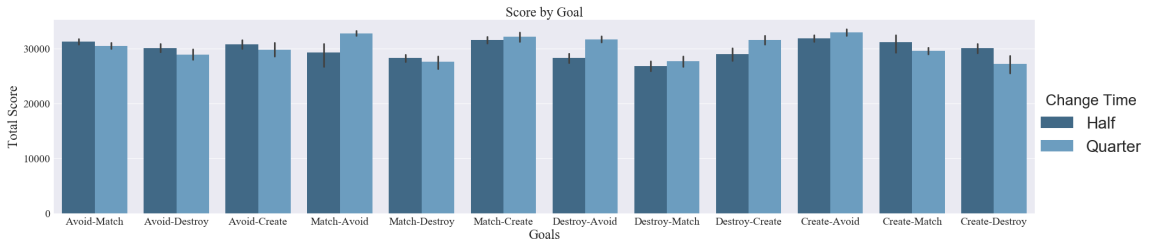

Figure 18. Average score for each of the 24 conditions. Black bar shows the 95% confidence interval for the score.

These results are consistent with the baseline results discussed in Section 3.2. The

baseline results showed that the line and collision avoidance agents performed best in the `avoid` and `create` goals, while the agents performed worse in the `match` and `destroy` goals. Thus, it is expected that any game with the `match` or `destroy` goal will have a lower final score.

The varying baseline performance of the line and collision avoidance agents across the goals explains the score difference between the two change times for the same goal combination. Because the goals are not present for equal time in the quarter change time scenario Goal-B has a larger affect on the final score compared to Goal-A. Thus, conditions with `avoid` or `create` as the second goal will have a higher score than conditions with `match` or `destroy` as Goal-B. For example, in the `match-avoid` condition, the average score in the quarter game change time is 3,527 points higher than in the half time goal change.

Next, scores are compared to the score results. Perfect in this context refers to the scenario where the goal agent instantly detects goal changes and adapts immediately. However, because the goal agent cannot instantly detect changes, the actual score will always be less than this ideal score. The ideal score for each condition was calculated using the mean baseline results from Section 3.2. Figure 19 shows the comparison of the perfect scores to actual score. The figure shows that actual scores follow the same trends as the corresponding baseline scores, but at a lower average score. On average the actual score is 9,047 points lower than the perfect score. With respect to the ideal score, the `match-create` conditions had the best performance with an average difference of 7,585 points. The `avoid-create` goal had the worst performance with an average difference of 10,714 points.

The 9,000 point difference between the perfect and actual scores is a byproduct of the exploration to learn the goal, and negative feedback required to recognize a goal change. A large number of potential points are lost during the exploration phase as

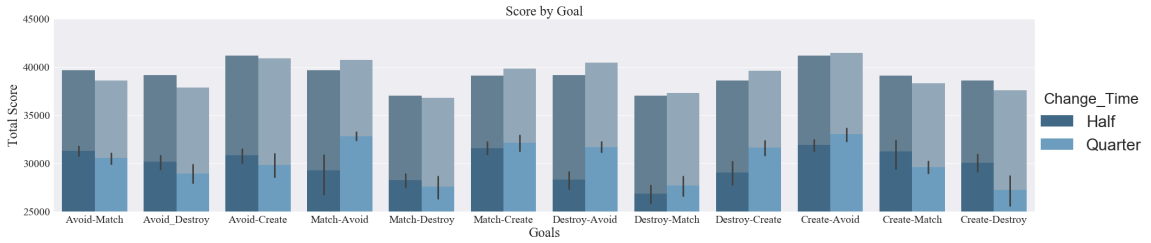

Figure 19. The difference in the perfect score for each condition compared to the actual score results.

a result of searching the environment for the goal. Additionally, for the goal agent to detect a change the agent requires some negative feedback which results in more lost points. The `match-create` goal has the best performance compared to its perfect score because the major negative scoring action, ship destruction, is a depth two mechanic meaning it will happen less often during the exploration phase.

Further reducing performance was the presence of false positive detections in games. False positives here are defined by the goal agent flagging a change when none had occurred. Each false positive triggers a period of exploration that further reduces the score. Across the 720 trials, there were a 101 total false positive detections in 95 games. The breakdown of false positive detections by goal is shown in Table 7. These most often occurred in the `destroy` goal state. A false positive was triggered 67 times across the `destroy` goal. At 23, `create` had the second highest number of false positive detections.

Table 7. Number of false positive detections by goal scenario and goal time.

# False Positives	`Avoid`	`Match`	`Destroy`	`Create`
Goal-A	2	3	20	13
Goal-B	0	0	47	10

4.2 Time Results

The next performance measure examined is detection time. Detection time is defined as the time it takes the goal agent to detect a change after the goal has change. Figure 20 shows the average detection time and 95% confidence interval for each condition. The key feature of this figure is the large `avoid-create` detection time result. In this goal scenario it takes the goal agent on average 2 minutes to detect a goal change.

Figure 20. The average time to detect the goal change in each of the conditions.

To gather more information the time results must be filtered to remove outliers. Detection times longer than 45 seconds can be reasonably assumed to be a pseudo false positive that has been triggered by means other then genuine change detection. Figure 21 shows the filtered detection time results where the long detection times have been removed. In this figure the detect time range has been greatly reduced making comparisons across conditions easier.

Figure 21. The average time taken to detect the goal change in each of the conditions with outliers removed.

The `match-destroy` conditions have best detection time, taking an average of 4.99

seconds to detect the goal change. The `avoid-create` condition continues to be the worst performer, taking an average of 35.1 seconds to detect a change. Across all goals the average detection time is 9.19 seconds. Lastly this chart shows that detection time is not affected by change time. As seen in Table 8 there is no significant correlation between the change time and the detection time. This proves the null hypothesis that the time a goal change occurs effect the agents ability to detect the change.

Not captured in either Figure 20 or 21 are the number of games without any detections. In 18 games no goal change was detected. This was a serious issue in the `avoid-create` condition. Of the 60 games in this condition no change was detected in 14 of them. In addition to those, a change was detected over 45 seconds after the goal switch in 30 other trials. That results in a change detection between (0, 45] seconds in only 26% of the `avoid-create` trials. Additionally the detection time in the 26% of proper detections has the worst performance across all conditions.

Further investigation of the `avoid-create` condition was conducted to determine the cause of the longer detection time. In each goal scenario, when the goal changes ships continue to pursue their original Goal-A targets, as the goal agent is unaware of the change. When they encounter their intended target that is when the goal agent receives feedback that a change has occurred and triggers a period of exploration to discover the change. For example, in `avoid-destroy` the goal agent only notices the change after it sees that ships are receiving -150 for landing on planets as opposed to the predicted 150 points. In the `avoid-create` scenario, however, ships cannot generate the necessary feedback. When the goal changes from `avoid` to `create`, all of the base colored planets are removed from the screen and replaced with the fusion

Table 8. Correlation between change and detection times.

Correlation	Pearson's Coefficient (r)	P-Value
Change Time x Detection Time	-0.0166	0.681

color planets. However, because the primary ships on screen are still targeting their home planets they fly off screen where they are deleted without sending any feedback to the goal arbiter. Then, new ships that spawn have to wait 20 seconds for the time delayed exploration to occur. If a change is not detected before the goal arbiter begins learning from the time delayed exploration, no change may be detected.

These results show that both the goals being switched to and from affect detection time.

4.3 Goal Similarity

Next the detection time was compared to the relative similarity of each goal to determine how the similarity of goals affected detection. Table 10, shows the point difference between the positive scoring actions of Goal-A to the corresponding points for the action in Goal-B. For example, if `avoid` were Goal-A, a ship receives 150 points for landing on a planet, if `match` were Goal-B, then a ship instead scores 50 points for landing on a planet. This results in a 100 point difference between the goals. The lower the number the more similar the goals are. Similarity was calculated using only the positive score actions because those are the actions that the goal agent will be assigning during steady state. The goal agent will detect goal drift based on changes to the positive score actions because those are the actions it will be taking. The assumption is that negative score actions will not be taken after exploration, thus the goal agent will not be detecting change based off them. Then, when comparing the similarity of goals it is most important to look at how the positive score actions change from Goal-A to Goal-B.

The `create` goal creates a special exception due to the mechanics of *Space Navigator*. Because the planets swap between `create` and the other three goals, the planet action is not applicable to either the primary or fusion ships. Thus, when going to or

Table 9. Relative point similarity of goals based on positive scoring actions. Higher values indicate less similar goals.

Goal A \ Goal B	Avoid	Match	Destroy	Create
Avoid	-	100	300	0
Match	250	-	250	150
Destroy	300	300	-	300
Create	150	175	250	-

from `create`, the planet action is not taken into account when calculating goal similarity. This causes a 0-point difference from `avoid` to `create` because the ships do not have the opportunity to test landing on a planet due to the colors changing. This is in line with the `avoid-create` behavior described earlier. In relation to the positive score action the two actions are essentially the same due to the agent's inability to get feedback.

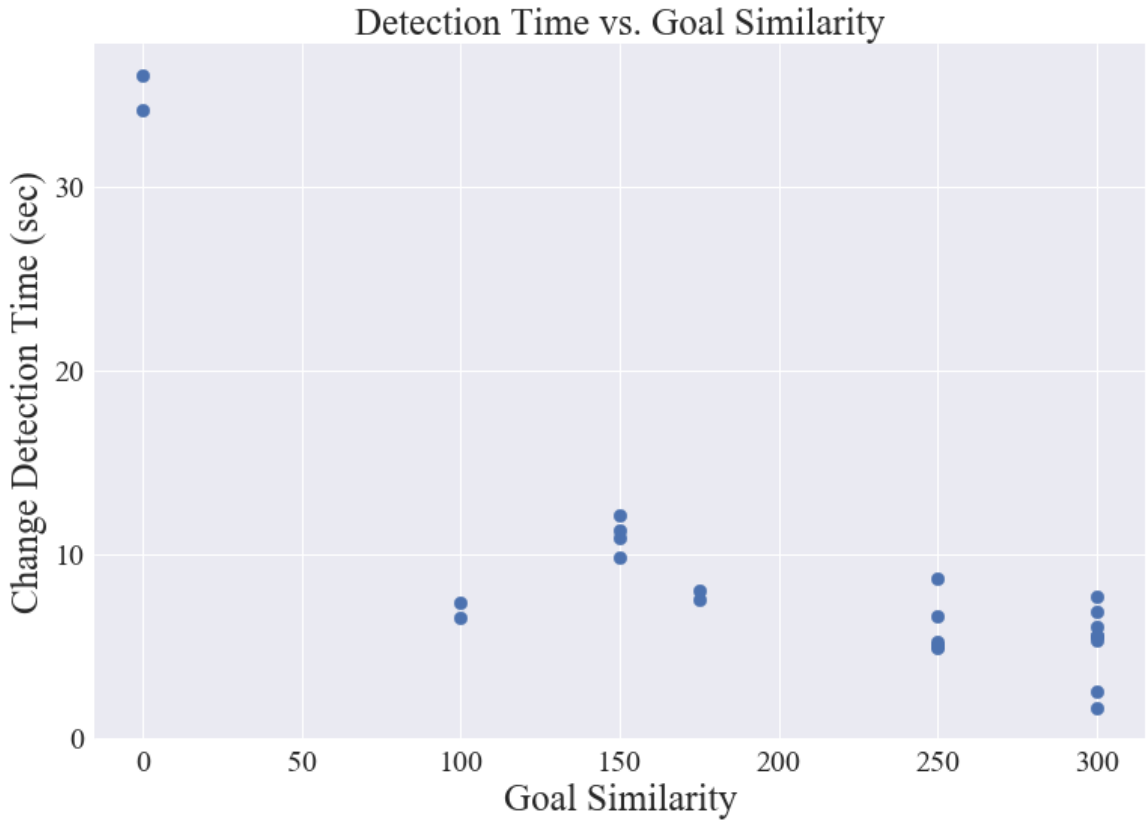

Figure 22. Detection time versus goal similarity.

Table 10. Correlation between goal similarity and detection time.

Correlation	Pearson's Coefficient (r)	P-Value
Goal Similarity x Detection Time	-0.5079	<0.001

Comparing the goal similarity to the detection time results shows a strong correlation between the two variables. Table 10 shows relation is significant beyond the $p \leq 0.05$ level. The correlation can be seen visually in Figure 22. The detection for the three conditions across each of the starting goals closely matches the similarity between the goals. From the `avoid` goal, `destroy` is the fastest to detect, `match` is second and `create` is third. In `destroy` no goal is fastest to detect with each of the three conditions all having overlapping confidence intervals. The `create` goal is the goal that does not follow this logic. Based on similarity `create-destroy` should be the fastest to detect, but that is not matched in the data, with it being only 0.1 seconds faster on average than `create-match`.

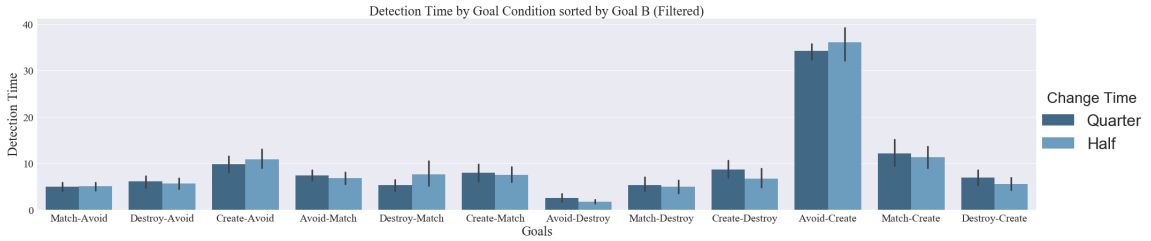

Figure 23. The average time taken to detect the goal change in each of the conditions with detections >45 seconds removed, organized by Goal-B.

For example, we would expect a Goal-B `match` to be detected earliest when `destroy` is Goal-A, but the confidence intervals for each Goal-B `match` condition are overlapping. It holds true for the other three goals however.

One interesting discovery is that there is no correlation between the time it takes to detect a change and the difference between expect vs actual score. We would expect that the longer it takes to detect a change, the greater the difference in expected versus actual score. That is not the behavior seen in Figure 24. The figure shows the

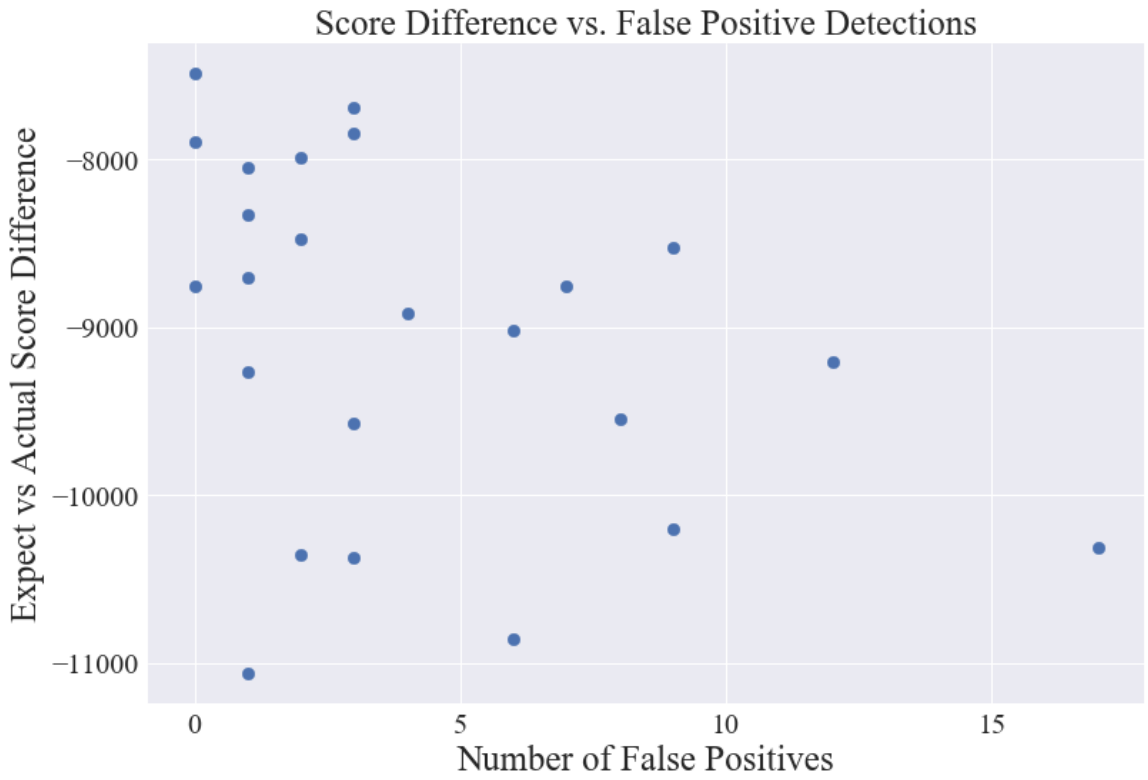

Figure 24. Detection time vs. difference between expected and actual score.

detection time for each of the 24 conditions compared to the difference in expected versus actual score. There is no correlation between the two metrics. Some conditions with longer detection times have a smaller point difference. Thus, the score difference is truly a product of the exploration and not detection time.

4.4 Chapter Summary

This chapter outlined the results from the experiment described in Chapter 3. The experimental results show that the goal agent is able to detect goal changes and adapt when the change occurs. The detection time does depend on both the goal being switched too and the goal being switched from. The similarity of the goals is a highly significant factor in the time it takes to detect a goal. The less similar the goals, the faster change detection is.

V. Experiment 2

This chapter details the methodology and results of the follow-on experiment that was conducted after analyzing the results from the first experiment. The second experiment aims to resolve several of the shortcomings of the first experiment to make the results more comparable to human performance. Section 1 details the methodology for the second experiment. Section 2 outlines the results and compares them to the results obtained from Experiment 1. The chapter concludes with a discussion on the importance of the new results and how they can be used in future work.

5.1 Methodology

One curious result from the first experiment is the `avoid-create` detection time. To a human, this change would be the easiest to detect due to the visual transformation of the planets. Once the primary and fusion colored planets swap, the human would know that an accompanying goal change has also occurred. Yet for the goal agent it is the scenario that takes the longest to detect the change. The human has a clear advantage of being able to use the visual feedback of the environment to detect a change. In order to obtain results that are more comparable to human performance, the goal agent needs to be updated with expanded perception capabilities.

Specifically, changes were made to the goal arbiter's target assignment perceptions. Because the goal arbiter does not have the ability to directly monitor the planets' locations, when the planets swap for the `create` goal no feedback is obtained. The goal agent would simply update its memory to ensure that off-screen planets were no longer passed as targets. To increase the agent's capabilities, feedback was added such that when an off-screen planet was passed as a target, a large error value is

passed to the change detection module. Thus, when several off screen targets would be passed, the change detection flag is triggered and exploration begins.

The hypothesis for this experiment is that this system will greatly improve the detection time for the `avoid-create` scenario and to a lesser extent improve the `match-create`, `create-avoid`, `create-match`, and `create-destroy` scenarios. The system will not improve the `destroy-create` detection time because at no point in the destroy goal are ships directed toward planets, thus the error is never passed indicating to the goal agent that planets have moved off screen. Finally, this change should have no effect on any scenario where the `create` goal is not present.

For the second experiment only games with the half game change time were run. This reduced the number of conditions from 24 to 12. This change was made to reduce the simulation run time and remove unnecessary data. The results from experiment 1 showed that there is no correlation between change time and detect time. The goal change time only affects the score results due to varying performance across the four goals. This thesis is primarily focused on the effects different goals have on detection rather than overall score performance. Thus, having two different change times yields redundant information. All comparisons made with Experiment 1 will be made using only the half game change times as to keep the comparisons valid.

5.2 Results

The detection time results, seen in Figure 25, show that the change to the goal arbiter greatly improved the detection time in the `avoid-create` scenario. The average detection time was reduced from 36 second down to 5.6 seconds. All 30 `avoid-create` games had proper detections compared to only 8 in Experiment 1. Additionally, as hypothesized the `match-create`, `create-avoid`, `create-match`, and `create-destroy` detection times were reduced. The `destroy-create` scenario was the only that had

an increase in average detection time.

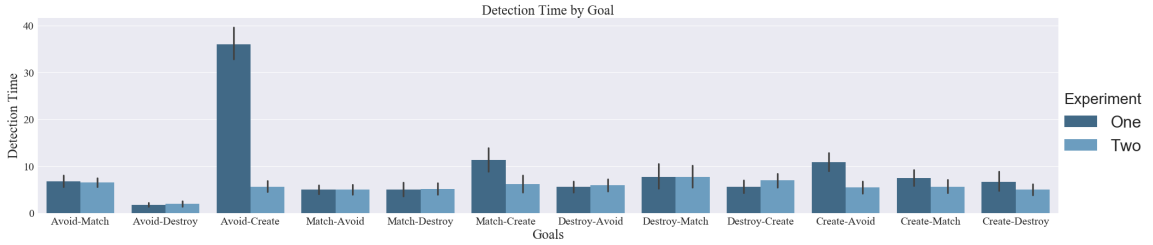

Figure 25. Comparison of detection time between Experiment 1 and Experiment 2.

Table 11 shows the statistical significance from the comparisons of means using a t-test between the two experiments. The difference in means is significant for the `avoid-create`, `match-create`, and `create-avoid` scenarios. It is significant for the `create-match` scenario. Lastly the change for both the `destroy-create` and `create-destroy` scenarios was not significant. These results show that change did have the intended effect of improving detection time.

Despite the improvements to detection time, the overall score performance did not have a corresponding increase. At 29,920 points, the average score across all games from Experiment 2 is only marginally higher than the average score of 29,888 for Experiment 1. This indicates that, although a change was not being explicitly detected in Experiment 1, the goal agent was still learning the new goal. Both experiments had a similar number of false positive detections which could be what caused the similar scores. Reducing the false positive detections is likely the best way

Table 11. Significance test for the detection means between experiment 1 and 2. (* denotes significance at the 0.05 level.)

Scenario	DF	F-Statistic	P-Value
avoid-create	36	-20.099	<0.001*
match-create	58	-3.233	0.002*
destroy-create	53	0.777	0.440
create-avoid	57	-4.543	<0.001*
create-match	56	-1.746	0.086
create-destroy	57	-1.359	0.179

to improve overall score performance.

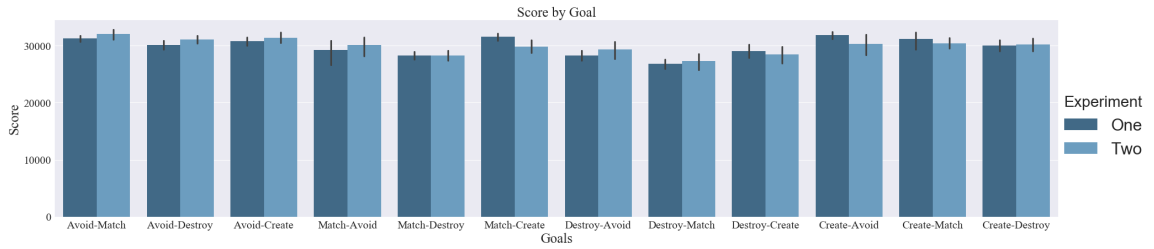

Figure 26. Comparison of score results between Experiment 1 and Experiment 2.

The results of this experiment are significant because they provide a new baseline for comparison to human performance. The change to the goal agents fix the detection shortcoming found in Experiment 1. The feedback from off-screen planets mirrors a humans capability to visually detect the color change of the planets. The Experiment 2 results provide a much fairer comparison to the human performance due to the goal agent change. The change to the goal agent increases it's performance without giving it specific knowledge of the goal scenarios.

5.3 Chapter Summary

This chapter outlined the results from the second experiment. The goal agent was given increased capabilities to detect when planets left the screen. This mimics a humans ability to recognize the visual change. The detection time results improved significantly for three of the goal scenarios. The increased detection time performance did not correlate with a similar increase in score. The results provide a baseline of results that will be more comparable to human performance on the same task.

VI. Human-Subject Experiment

A human subject experiment has been designed to test the human's change detection capabilities at the same *Space Navigator* task as the agents, which will help to identify the categories of goal changes that each group excels at. The primary goal of the experiment is to determine if there is a difference between human and agent performance. Additionally, it will look at how humans detect change within the environment. This chapter first outlines the experimental design, then explains the data to be collected during the experiment. As of writing, the outlined experiment has not been conducted. The hope is the the experiment can be run as part of the future work for this research.

6.1 Experimental Design

In the human-subject experiment, participants will complete the same task as the autonomous agents in the previously described experiments. The human will take the place of the agents to compare goal detection between the two groups. The objective is to keep the task as similar as possible to what the goal agent experienced during Experiment 2. This allows the measurements to remain comparable between the two groups.

First, each participant is taught how to play *Space Navigator* by demonstrating the various mechanics for the ships. For each of the mechanics: landing on a planet, shield depletion, ship destruction, and ship fusion, participants are shown a short video of how the mechanic functions. After being taught how to play the game, participants engage in a five minute practice round of *Space Navigator*. The practice round is intended simply to familiarize the subject with the mechanics of the game. No points are rewarded for any of the actions. This allows the player practice the

various ship interactions without biasing them toward any one action.

After completion of the practice round, participants complete six games with a goal change. The number of rounds is decreased from twelve in the automated experiment to six for the human-subject experiment, in order to prevent player fatigue and to limit the number of redundant goals seen by the player. If tested on all twelve goal change conditions, the player would likely be able to learn each goal and quickly identify each change by the end of the experiment. The goal agent, however, starts with no knowledge of the goals, so games where the human is still learning are more comparable. Each goal will be seen three times across the six rounds. Additionally, the games are shortened to 5-minutes to reduce player fatigue.

Because six different rounds would require 720 participants to test all combinations, blocking is used to confound the learning effects of experiencing the goal changes multiple times. Randomized complete blocking with 24 participants is used to determine the testing order. The test schedule for the twenty-four participants is shown in Table 12.

Within each round, the goal changes at a random time between 105 to 195 seconds into the game, or $\pm$ 45 seconds from the midpoint of the game. After the participant notices the goal change, they are asked to write down the game time at which they noticed the change. Subjects are told beforehand that only one goal change will occur during each game. After each round the participant then writes down what they thought each of the goals were. While players will be told to try and achieve the highest score possible, this experiment is primarily interested in the human's change detection capabilities. It is highly unlikely that any human player will be able to perform to the same level as the line and collision avoidance agents, making final score an unfair comparison between the two groups.

Table 12. Testing schedule for human subject experiment. A, C, D, M are acronyms for the `avoid`, *create*, *destroy*, and *match* goals.

User	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6
1	M-D	C-A	A-D	A-M	C-M	D-C
2	C-M	C-A	A-D	M-D	A-M	D-C
3	C-A	M-D	D-C	C-M	A-M	A-D
4	D-C	C-M	M-D	A-M	A-D	C-A
5	A-M	M-D	C-A	C-M	D-C	A-D
6	C-M	C-A	A-M	A-D	M-D	D-C
7	A-D	M-D	A-M	D-C	C-M	C-A
8	M-D	C-M	A-M	D-C	A-D	C-A
9	C-M	M-D	C-A	A-M	D-C	A-D
10	A-M	C-A	C-M	D-C	M-D	A-D
11	A-M	A-D	C-M	C-A	D-C	M-D
12	D-C	A-M	C-M	A-D	C-A	M-D
13	D-C	A-D	M-D	C-A	A-M	C-M
14	C-M	A-D	A-M	M-D	D-C	C-A
15	C-A	M-D	A-M	C-M	A-D	D-C
16	D-C	A-D	A-M	M-D	C-M	C-A
17	A-M	A-D	D-C	C-M	M-D	C-A
18	A-M	A-D	C-A	D-C	C-M	M-D
19	M-D	A-M	D-C	C-M	C-A	A-D
20	C-A	M-D	D-C	A-M	A-D	C-M
21	C-M	A-D	A-M	D-C	C-A	M-D
22	A-D	C-A	D-C	C-M	M-D	A-M
23	A-M	M-D	A-D	C-A	D-C	C-M
24	A-M	C-M	C-A	A-D	D-C	M-D

6.2 Recorded Data

Before the experiment begins, participants are asked to complete a basic demographic questionnaire (Appendix A). Of particular interest on this questionnaire is the subject's prior experience with *Space Navigator* and their general gameplay exposure. Prior experience with *Space Navigator* could negatively affect the subjects' performance, as previous experiments using the environment only had one goal. The subject may be biased towards the `avoid` goal, because that is the goal they know from previous experiments with the game. General gameplay exposure will likely increase a subject's ability to detect change, as many of the goal changes are similar to what might be encountered in other video games.

During each trial the participant is asked to note down when they notice a goal change. Additionally at the end of each trial, the participant is asked to write down what they thought the starting and ending goals were. This primarily is used to determine how long the participant took to detect the goal change. Additionally, it will reveal if the participant has fully learned the goals or is only accomplishing one layer of the goal.

In addition to the questionnaire, participants will be recorded during the experiment and be asked to comment on what they are doing during the game. This will provide greater insight into how the subject is noticing goal changes and the variations they are making to their behavior in response. The recording will also allow for the subject to identify when they notice a change, but have not fully confirmed it.

After the participants complete all of the games, they fill out a final questionnaire (Appendix B) where they are asked about how they detected the goal changes and which goal they thought was the hardest to detect. Additionally, are asked to complete a personality test to determine if certain personality traits are more conducive to detecting change over others.

Gameplay data is recorded from within *Space Navigator*. The gameplay data include:

- *Basic game data*, including the round identifier, final score, goal scenario, and change time.
- *Player action data*, containing the information on all trajectories drawn by the player.
- *Ship data*, containing all ship related events. This includes: spawn time, collisions, fusions, planet collisions, and screen exits.

The gameplay data provides objective measures for the participants behavior. The data will be used to determine how a player responds to changes. Mapping the players points over time will inform how quickly the player is learning the goal. Trajectory data will be used to determine how play changes near the goal change. Measures such as trajectory draws per second, number of trajectory redraws, and length of time a ship is on screen before a trajectory is drawn can be used to determine the impact of a goal change beyond just score.

6.3 Expected Results

The expected results from this experiment are that the human will perform significantly worse than the goal agent at the beginning of the experiment, but improve and potentially surpass the goal agent with practice. For the first several trials the participant will likely have a difficult time detecting the change due in inexperience with the task. Each new goal the participants encounters will also make it harder for the participant to detect a change. Lack of experience with the goal will make it harder to determine if a change has occurred. For these trials, it is expected that the goal agent will greatly outperform the human.

The experiment will likely have a large learning effect. For later trials the participant will have experienced each of the goals and will be able to identify each more quickly. Additionally, the participant will have a better idea of what they need to do to identify the goal change. By the end of the experiment it is likely that the human will perform at the same level as the goal agent. For scenarios with the `create` goal, it is likely that the human will be able to instantly detect a change due to the visual alteration of the planets swapping.

6.4 Chapter Summary

This chapter presented a human-subject experiment to complement the results detailed in Chapter 5. A human-subject experiment will provide valuable information on the differences between agent and human goal change detection. This data could ultimately be used in the creation of a human-machine team proficient at identifying goal changes. The hope is this experiment will be run after the completion of this thesis to further extend this research.

VII. Conclusion

This chapter summarizes the research question, methodology used to answer the research and investigative questions, and the contributions of this research. The chapter concludes with a look at limitations of the research and potential future work that can expand on the results of this experiment.

7.1 Summary of Research Question

This research answered the question of whether different categories of goals affect an agents ability to detect and adapt to goal changes. Investigative questions answered: how goals can be defined within a gameplay environment, how change can be detected by an agent, and how does the order of goals affect detection?

The answer to these questions can help inform the development of goal detection agents within simulated environments. This research provides additional insight in to how agents can goal detect changes and how the goals affect detection performance. This is a vital step in developing autonomous agents for highly dynamic environments. Additionally this research will aid in the development of agents for human machine team, that can compensate for the humans weakness and inform the human as to potential goal changes within the environment.

7.2 Summary of Methodology

This research utilized the tablet-based computer game *Space Navigator* to simulate a dynamic goal environment. In order to test the research question a new set of goals were defined and added to *Space Navigator* based on Djaouti et al.'s [1] classification of goal mechanics within games. The four goals, `avoid`, `match`, `destroy`, and `create`, were implemented into the environment. In addition to the goals, new game mechanics

were added giving ships a shield to survive single collisions and a fusion mechanic to form new ship colors.

Fully autonomous play of *Space Navigator* was achieved with new line and collision avoidance agents that improved upon those previously developed for the environment. The line agent utilizes the A* path finding algorithm to draw intelligent trajectories for the ships that avoid other objects in the environment.The collision avoidance agent uses raycasting to detect potential collisions and a step back behavior to avoid collisions. The agents were tested across the four new goals within the environment to baseline their performance in each goals.

A goal detection agent was created to learn the current goal and detect when a goal change occurs. The goal agent used concept drift detection to detect change within the environment. Two sliding windows track the prediction error for ship actions. When the error passes the threshold a flag is triggered that a goal change has occurred. When a change is detected by the agent it begins to explore the environment to learn the goal. In addition to goal detection the agent is responsible for passing target objects to the ships.

The experimental portion of this research examined the performance of the goal agent across 24 different goal change scenarios. Each of the 12 possible 2-goal scenarios were tested with two different change times. Each condition was run 30 times. After data collection, statistical analysis was done to determine the effects the goals had on performance and detection. After the initial experiment analysis, new research questions were raised and addition experiments were run.

7.3 Summary of Contributions

The first contribution of this research is the proposed taxonomy of change within a gameplay environment. This taxonomy helps formalize how changes effect an en-

vironment by listing the three areas of a game where changes can be made and how the changes can be made. For a game like *Space Navigator*, which has undergone numerous changes across research efforts, it is important to be able to formally describe how new changes to the game affect the underlying structure of the game. This helps to ensure that changes are accomplishing their intended purpose. This taxonomy can be utilized for analyzing other environments that many be used for change detection research.

The second contribution of this research is the development of new agent for the *Space Navigator* environment. These agents can be utilized for future research and help to make *Space Navigator* a more robust testing environment.

Finally the experimentation results provide insight into goal detection. The results of the first experiment showed that detection is affected by the goal being switch from. Similarity of the goals had a statistically significant affect on detection time. The more similar the goals were, the longer it took the goal agent to detect the goal change. The results from he `avoid-create` conditions demonstrated the limitation of detecting change purely on feedback detection. In Experiment 2, adding further detection capabilities to the goal agent based on planet changes showed that detection time could be further reduced. This suggests that a goal detection agent should be provided with more perception to increase detection capabilities.

7.4 Future Work

The results of this thesis present many avenues for future research in the area of goal detection. The following are suggestions of future work that can be done to follow up and extend this research:

- Compare results to human performance: The results from this research only inform how the autonomous agent is affected by different categories of goal

changes. Running the human-subject experiment detailed in Chapter 6 could provide a wealth of new data to explore. Analysis of the differences between human and agent performance is vital for the creation of effective human machine teams that can work in adaptive goal environments.

- Test goal agent in a human-machine team: An additional human subject experiment can test how a human would perform when paired with the goal agent. This experiment would aim to answer the question of whether a human machine team between a human and the goal agent would perform better than an individual human. In this setup, the goal agent would inform the player when it detects a goal change with the aim to increase the human's detection and reaction time.

- Repeat within a different environment: *Space Navigator* has a relatively limited set of mechanic which restricts the variety of goals available in the environment. The goals in this experiment were limited to the avoid and match mechanics with varying depth and breadth to form the four goal set. A new environment with additional play mechanics opens up the possibility for different types of goals to be tested. This experiment would also serve to validate the function of the goal agent within another environment.

- Test harder goal scenarios: For this research all ships shared the same goal, and each goal had a max depth complexity of two. Testing the agent under more complex goal scenarios could reveal further information about detection. Having individual goals per ship color will increase the complexity of detecting changes. Additionally, goals with greater depth may make detection and adaption harder.

- New methods for switching between exploration and exploitation: The goal agent currently explores for a fixed length of time before switching to exploita-

tion. While a new switching method would have no effect on detection time, it would increase overall performance within the environment. Other methods of switching could be used to exploit earlier, thus reducing the point loss of needless exploration. This would additionally help reduce the impact of false positive detections on the total score.

Appendix A. Pre-Experiment Questionnaire

1. Participant Number (Assigned By Researcher):
2. Name:
3. Email Address:
4. Age:
5. Handedness: Left Right Ambidextrous
6. Gender: Male Female
7. Number of previous Space Navigator studies you have participated in:
8. How much experience do you have with the following:

In a given year, on how many days do you interact with the following types of devices?

	Never	<1 per month	1-3 per month	1-2 per month	3-6 per week	Daily
Laptop Computer	0	1	2	3	4	5
Tablet computer	0	1	2	3	4	5
Smart phone	0	1	2	3	4	5
Desktop computer	0	1	2	3	4	5
Gaming consoles	0	1	2	3	4	5

In a given year, on how many days do you interact with the following types of devices?

	Never	<1 per month	1-3 per month	1-2 per month	3-6 per week	Daily
Any Kind	0	1	2	3	4	5
Simulation (SimCity)	0	1	2	3	4	5
Battle-Arena (LoL, Overwatch)	0	1	2	3	4	5
Role-Playing (WoW)	0	1	2	3	4	5
Action (Mario, Donky Kong)	0	1	2	3	4	5
First Person Shooter (Halo)	0	1	2	3	4	5
Strategy (Civilization)	0	1	2	3	4	5
Puzzle (Tetris)	0	1	2	3	4	5
Casual (Angry Birds)	0	1	2	3	4	5
Music (Guitar Hero)	0	1	2	3	4	5
Sports (Madden Football)	0	1	2	3	4	5
Board (Ticket to Ride)	0	1	2	3	4	5
Card (Poker, Pinochle)	0	1	2	3	4	5

Appendix B. Post-Experiment Questionnaire

1. Participant Number (Assigned By Researcher):
2. What method were you using to identify goal changes?

3. What goal was the easiest to identify?

4. What goal was the hardest to identify?

On the following pages, there are phrases describing people's behaviors. Please use the rating scale below to describe how accurately each statement describes you. Describe yourself as you generally are now, not as you wish to be in the future. Describe yourself as you honestly see yourself, in relation to other people you know of the same sex as you are, and roughly your same age. So that you can describe yourself in an honest manner, your responses will be kept in absolute confidence. Please read each statement carefully, and then fill in the bubble that corresponds to your reply.

	Very Inaccurate	Moderately Inaccurate	Neither Inaccurate nor Accurate	Moderately Accurate	Very Accurate
1. Feel comfortable around people	☐	☐	☐	☐	☐
2. Keep in the background	☐	☐	☐	☐	☐
3. Have frequent mood swings	☐	☐	☐	☐	☐
4. Carry out my plans	☐	☐	☐	☐	☐
5. Suspect hidden motives in others	☐	☐	☐	☐	☐
6. Have a vivid imagination	☐	☐	☐	☐	☐
7. Enjoy hearing new ideas	☐	☐	☐	☐	☐
8. Pay attention to details	☐	☐	☐	☐	☐
9. Am not interested in abstract ideas	☐	☐	☐	☐	☐
10. Make people feel at ease	☐	☐	☐	☐	☐

	Very Inaccurate	Moderately Inaccurate	Neither Inaccurate nor Accurate	Moderately Accurate	Very Accurate
11. Get back at others	☐	☐	☐	☐	☐
12. Often feel blue	☐	☐	☐	☐	☐
13. Know how to captivate people	☐	☐	☐	☐	☐
14. Do not like art	☐	☐	☐	☐	☐
15. Carry the conversation to a higher level	☐	☐	☐	☐	☐
16. Rarely get irritated	☐	☐	☐	☐	☐
17. Seldom feel blue	☐	☐	☐	☐	☐
18. Find it difficult to get down to work	☐	☐	☐	☐	☐
19. Would describe my experiences as somewhat dull	☐	☐	☐	☐	☐
20. Do not enjoy going to art museums	☐	☐	☐	☐	☐

	Very Inaccurate	Moderately Inaccurate	Neither Inaccurate nor Accurate	Moderately Accurate	Very Accurate
21. Am skilled in handling social situations	☐	☐	☐	☐	☐
22. Dislike myself	☐	☐	☐	☐	☐
23. Make friends easily	☐	☐	☐	☐	☐
24. Panic easily	☐	☐	☐	☐	☐
25. Make plans and stick to them	☐	☐	☐	☐	☐
26. Don't talk a lot	☐	☐	☐	☐	☐
27. Am always prepared	☐	☐	☐	☐	☐
28. Get chores done right away	☐	☐	☐	☐	☐
29. Tend to vote for conservative political candidates	☐	☐	☐	☐	☐
30. Insult people	☐	☐	☐	☐	☐

	Very Inaccurate	Moderately Inaccurate	Neither Inaccurate nor Accurate	Moderately Accurate	Very Accurate
31. Am very pleased with myself	☐	☐	☐	☐	☐
32. Have a good word for everyone	☐	☐	☐	☐	☐
33. Believe that others have good intentions	☐	☐	☐	☐	☐
34. Don't like to draw attention to myself	☐	☐	☐	☐	☐
35. Am not easily bothered by things	☐	☐	☐	☐	☐
36. Do just enough work to get by	☐	☐	☐	☐	☐
37. Have a sharp tongue	☐	☐	☐	☐	☐
38. Have little to say	☐	☐	☐	☐	☐
39. Shirk my duties	☐	☐	☐	☐	☐
40. Tend to vote for liberal political candidates	☐	☐	☐	☐	☐

	Very Inaccurate	Moderately Inaccurate	Neither Inaccurate nor Accurate	Moderately Accurate	Very Accurate
41. Accept people as they are	☐	☐	☐	☐	☐
42. Respect others	☐	☐	☐	☐	☐
43. Am the life of the party	☐	☐	☐	☐	☐
44. Cut others to pieces	☐	☐	☐	☐	☐
45. Avoid philosophical discussions	☐	☐	☐	☐	☐
46. Feel comfortable with myself	☐	☐	☐	☐	☐
47. Believe in the importance of art	☐	☐	☐	☐	☐
48. Don't see things through	☐	☐	☐	☐	☐
49. Waste my time	☐	☐	☐	☐	☐
50. Am often down in the dumps	☐	☐	☐	☐	☐

Bibliography

1. A Gameplay Definition through Videogame Classification. *International Journal of Computer Games Technology*, pages 1–7, 2008.

2. James Anhalt, Alexander Kring, and Nathan Sturtevant. Ai navigation: It's not a solved problem - yet. Game Developers Conference, 2011.

3. Sylvester Arnab, Theodore Lim, Maira B Carvalho, Francesco Bellotti, Sara Freitas, Sandy Louchart, Neil Suttie, Riccardo Berta, and Alessandro De Gloria. Mapping learning and game mechanics for serious games analysis. *British Journal of Educational Technology*, 46(2):391–411, 2015.

4. Eletronics Arts. Dragon age: Origins. [CD-ROM], 2009.

5. Jason Bindewald. Adaptive automation design and implimentation, 2015.

6. Mica R Endsley. Autonomous horizons: System autonomy in the air force: A path to the future. *Washington, DC: US Air Force Office of the Chief Scientist*, 2015.

7. Blizzard Entertainment. Starcraft. [CD-ROM], 1998.

8. João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.

9. PopCap Games. Bejeweled. [Windows], 2001.

10. Chris Garnick. A study of human reliance on imperfect automation, 2017.

11. Tyler Goodman, Michael E Miller, Christina F Rusnock, and Jason Bindewald. Timing within human-agent interaction and its effects on team performance and human behavior. In *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2016 IEEE International Multi-Disciplinary Conference on*, pages 35–41. IEEE, 2016.

12. Aaron Granberg. A* pathfinding project, 2017.

13. T Ryan Hoens, Robi Polikar, and Nitesh V Chawla. Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence*, 1(1):89–101, 2012.

14. Kevin Anthony Hoff and Masooda Bashir. Trust in automation: Integrating empirical evidence on factors that influence trust. *Human Factors*, 57(3):407–434, 2015.

15. Robin Hunicke, Marc LeBlanc, and Robert Zubek. MDA: A Formal Approach to Game Design and Game Research. *Workshop on Challenges in Game AI*, pages 1–4, 2004.

16. Kevin HWynee and Joseph Lyons. An integrative model of autonomous agent teammate likeness. *Theoretical Isssues in Ergonomics Science*, 2016.

17. Mark G Kelly, David J Hand, and Niall M Adams. The impact of changing populations on classifier performance. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 367–371. ACM, 1999.

18. Daniel King, Paul Delfabbro, and Mark Griffiths. Video game structural characteristics: A new psychological taxonomy. *International Journal of Mental Health and Addiction*, 8(1):90–106, 2010.

19. Jan.H.G Klabbers. The Gaming Landscape: a Taxonomy for Classifying Games. *LEVEL UP: Digital Games Research Conference*, pages 54–67, 2003.

20. R Koster. A grammar of gameplay: game atoms: can games be diagrammed. In *Presentation at the Game Developers conference*, 2005.

21. Namco. Pac-man. [Arcade], 1980.

22. Jeff Orkin. Applying goal-oriented action planning to games. *AI Game Programming Wisdom*, 2:217–228, 2003.

23. Jeff Orkin. Three states and a plan: the ai of fear. In *Game Developers Conference*, volume 2006, page 4, 2006.

24. Lilia Rejeb, Zahia Guessoum, and Rym MHallah. The exploration-exploitation dilemma for adaptive agents. In *Proceedings of the Fifth European Workshop on Adaptive Agents and Multi-Agent Systems*, 2005.

25. Richard Rouse. *Game Design: Theory and Practice (2nd Edition)*. Jones Bartlett Learning, 2004.

26. Stuart Jonathan Russell and Peter Norvig. Artificial intelligence: a modern approach (3rd edition), 2009.

27. Katie Salen and Eric Zimmerman. Rules of Play: Game Design Fundamentals. 2004.

28. Miguel Sicart. Defining game mechanics. *Game Studies*, 8(2):1–14, 2008.

29. Aleksandrs Slivkins and Eli Upfal. Adapting to a stochastically changing environment: The dynamic multi-armed bandits problem. Technical report, Rapport technique, Technical Report CS-07-05, Brown University, 2007.

30. Jake Spuller. Analysis of how communication affects human teams in a dynamic game, 2017.

31. Sierra Studios. Counter strike. [CD-ROM], 1999.

32. Taito. Space invaders. [Arcade], 1978.

33. Unity Technologies. Physics.spherecast documentation, 2018.

34. Sebastian B Thrun and Knut Möller. Active exploration in dynamic environments. In *Advances in neural information processing systems*, pages 531–538, 1992.

35. Stewart W Wilson et al. Explore/exploit strategies in autonomy. In *From animals to animats 4: Proceedings of the 4th international conference on simulation of adaptive behavior*, pages 325–332, 1996.

36. Indrė Žliobaitė. Learning under concept drift: an overview. *arXiv preprint arXiv:1010.4784*, 2010.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
14 Jun 2018	Master's Thesis	Sept 2016 — Jun 2018

4. TITLE AND SUBTITLE

EFFECTS OF DYNAMIC GOALS
ON AGENT PERFORMANCE

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Nathan R. Ball

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Electrical and Computer Engineering (AFIT/ENG)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-18-J-003

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research investigates how different categories of goals affect autonomous change detection in a dynamic environment. In order to accomplish this goal, a set of autonomous agents were developed to perform within an environment with multiple possible goals. The agents perform the environmental task while monitoring for goal changes. The experiment tests the agents over a range of goal changes to determine how detection performance is affected by the different categories of goals.
Results show that detection is highly dependent on what goal is being switch to and from. The point similarity between goals is the most significant factor in evaluating the change detection time. An additional experiment improved upon the goal agent and demonstrated the importance of having the proper perception mechanics for feedback within the environment.

15. SUBJECT TERMS

LaTeX,Thesis

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Major Jason Bindewald, PhD, AFIT/ENG
U	U	U	UU	88	19b. TELEPHONE NUMBER *(include area code)* (312)-785-3636, x4614; jason.bindewald@afit.edu