USER MODELING TOOLS FOR VIRTUAL ARCHITECTURE

Raja Uppuluri, B.Tech

Problem in Lieu of Thesis Prepared for the Degree of

Master of Science

UNIVERSITY OF NORTH TEXAS

May 2003

APPROVED:

Karl Steiner, Major Professor
Robert P. Brazile, Graduate Coordinator of Computer Science Department
Krishna Kavi, Chairman of the Computer Science Department
C. Neal Tate, Dean of the Robert B. Toulouse School of Graduate Studies

Uppuluri, Raja, *User Modeling Tools for Virtual Architecture*. Master of Science (Computer Science), May 2003, 83 pp., references, 9 titles.

As the use of virtual environments is becoming more widespread, user needs are becoming a more significant part in those environments. In order to adapt to the needs of the user, a system should be able to infer user interests and goals. I developed an architecture for user modeling to understand users' interests in a virtual environment by monitoring their actions. In this paper, I discussed the architecture and the virtual environment that was created to test it. This architecture employs sensors to keep track of all the users' actions, data structures that can store a record of significant events that have occurred in the environment, and a rule base. The rule base continually monitors the data collected from the sensors, world state, and event history in order to update the user goal inferences. These inferences can then be used to modify the flow of events within a virtual environment.

TABLE OF CONTENTS

LIST OF ILLUSTATIONS

Figure Page

CHAPTER 1

INTRODUCTION

Virtual environments (VE) are becoming a more common approach to information presentation and user interaction. While VE can be an inclusive term for many different presentation and interaction technologies and techniques, I use the term VE to mean an interactive, 3D presentation of a computer-generated space. Once the almost exclusive domain of high-end research laboratories, recent advances in 3D hardware and software have made high-fidelity VEs practical on home computers as well as in game consoles, dramatically expanding the number of potential applications and potential users.

The accelerating improvements and developments in the field of VE open the doors to a wide variety of applications. Some of the many application areas for VE are education, business, architecture, science, medicine, robotics, military, tools for handicapped, flight simulators, art, entertainment, sports and fitness. “For those times when you'd like to say, “*You just have to be there*”, virtual environments can offer the solution.” [1] VEs can allow users to experience and to evaluate situations that may be impossible or impractical to experience in the real world. For example, users can enter VE simulations of hazardous situations without experienceing economical or human loss.

Many VEs adopt a simulation-based control approach. That is, the rules that govern when and how events occur in the VE are based primarily on attributes of the world and the objects within it. This is an appropriate solution when the success of the VE experience depends on close fidelity to a real world situation (or some other set of rules). For example, in a good flight simulator, events should occur based on user actions and the physical properties of the airplane.

However, the success of some VE experiences does not depend on simulation alone. For example, training systems may seek not only to present events according to simulation rules, but may strive to create learning situations that are appropriate to a given student and a given world state. Success in these environments depends not only on simulation rules, but also on awareness of the user – their location, their activities, and even their goals and intentions. The development and use of such information is referred to as user modeling. “User modeling allows the computer to share a context with the user, in principle allowing for more effective communication.”[2]

“One approach to user modeling is to record various aspects of user activities and virtual world state and use this information to make inferences regarding the user’s intentions.” [3] The specific information used depends on the nature of the VE, the tasks the VE is meant to support, and the relevant user information. The user model should be able to recognize important situations and

user needs and act accordingly. The user modeling tools, which I developed can do this.

This work is part of an integrated, intelligent virtual environment architecture (VEA) for presenting various scenarios being developed by the Interactive Media Library at the University of North Texas. This VEA contains a narrative engine, capable of coordinating the agents and objects in the world to follow a scenario out line; a presentation management component, responsible for managing the timing, placement and form of scenario events; and a user modeling component, capable of reasoning users' behavior, ascertaining users' goals and communicating these goals with other modules. The work described in this paper is part of the user modeling component and explains the architecture behind the development of user modeling tools.

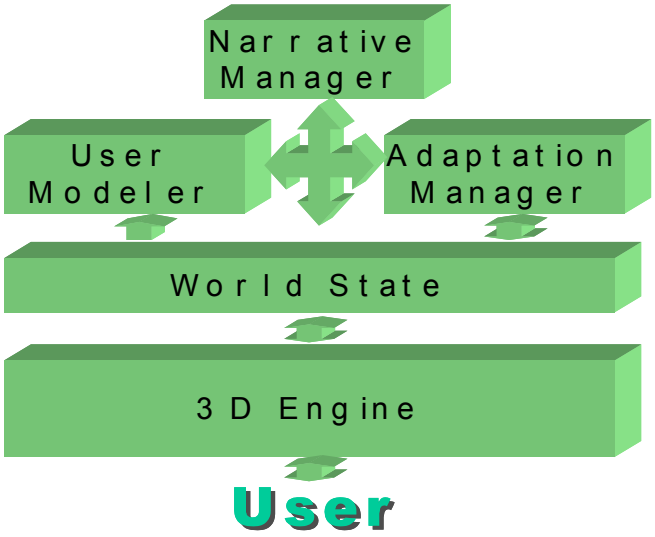

Fig (1)

CHAPTER 2

ARCHITECTURE

One of the important abilities of the proposed user modeling (UM) architecture is the inference of user goals. In order to achieve effective inferences, the architecture needs to be capable of acuisition, representation and maintenance of information about the user. The UM architecture includes three primary components: sensors, data structures and rule base. The components were implemented independently as separate but cooperative modules. The UM architecture contains various modules and the outline of the architecture is shown below.

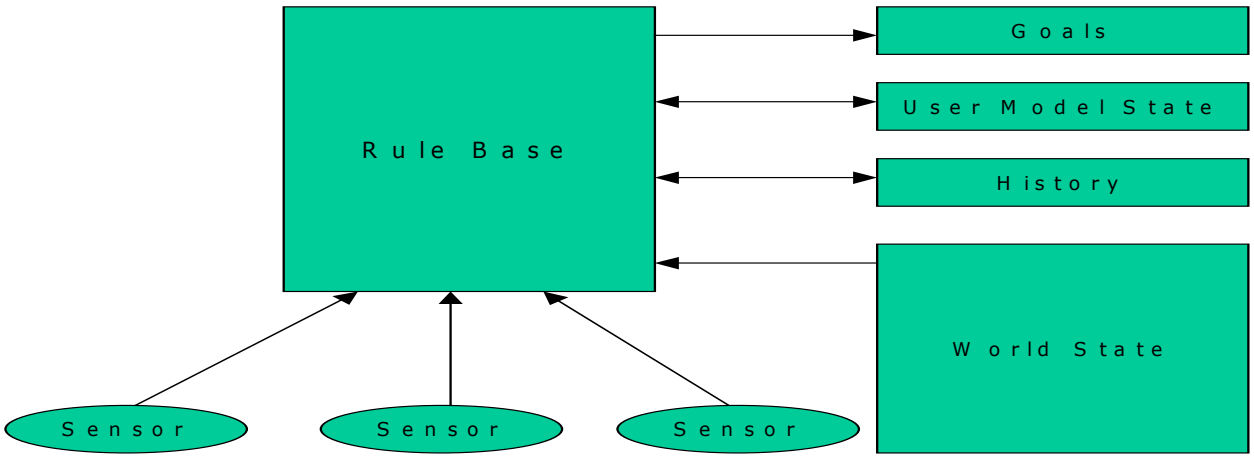

Fig (2)

The system components include sensors, data structures and a rule base. The sensors reports low level user and agent activities such as approaching objects or agents, interacting with objects or agents, etc. The data structures include a world state, history, and user model. The world state contains the information about objects or agents (e.g. location, potential uses, etc.). The history represents all the significant events that have occurred in the scenario. The user model contains the internal flags or facts used by the rule base. The rule base uses the information in these data structures to infer user goals.

Description of Test Environment

A test environment is needed to test the effectiveness of the proposed approach to user modeling. The environment would need to present a rich 3D world, include multiple goals that a user could choose to pursue at any given time, and include a variety of actions the user could take to pursue these goals. The success of the environment would also depend not just on simulation rules, but would also require knowledge of the users' goals. It was felt that a game-like environment would meet many of these requirements.

3D World

The simulated world is a large area containing natural features such as green fields, mountains, hills, and thick forests. Situated within this world are several areas that the user can explore - castles, villages, forts, and temples. The user can freely navigate this world and move from location to location.

Agents and Objects

The user encounters different agents and objects in the environment. The agents appear at different locations and provide various assistance or obstacles for the user. There are 5 agent types in the environment: monsters, samurais, hostile enemy soldiers, villagers, priests and a scorpion monster.

Objects are also located at various places in the environment. Each one of these has its specific purpose in the environment and the user can interact with these in different ways. The armor packs provides protection against attack and the health packs heal damage that was done to the player.

User Goals

Five goals that a user may pursue as part of their interaction with the environment have been identified. The goals are meant to be high level, and the goals may be achieved in a variety of ways. These goals are described in brief below:

Goal 1 – Build a new Civilization: User builds new towns.

Goal 2 – Destroy Enemies and Capture their Fort: User becomes a KNIGHT, assembles an army, and attacks a Fort.

Goal 3 – Be a Warrior: Use interacts with other agents in a hostile manner.

Goal 4 – Explore the World: User peacefully explores the environment.

Goal 5 – Complete a Quest: User becomes a KNIGHT and defeats SCORPION.

User Actions

There are three basic user interactions: “KILL”, “HIRE/HELP”, and “PICK.” KILL allows the user to attack and destroy another agent. HIRE/HELP allows the user to recruit another agent and receive their assistance. PICK allows the user to get and use objects such as the armor and health packs.

Approach to User Modeling

The components of the user modeling architecture are sensors, a knowledge base and a rule base.

Sensors

The purpose of the sensors is to keep track of user interactions with the different actors and objects in the environment. There are two types of sensors. One sensor performs a continuous scan around the user, detecting nearby agents and objects and enabling user-initiated interactions. The other type of sensor is similar but attached to agents, allowing them to initiate actions under certain circumstances.

Knowledge Base

Recording a history of the activities inside the environment generates a significant amount of information; however, this information is necessary to infer the user interests and goals. The knowledge base contains different varieties of data including the information about what the user is currently doing, what the user did before, and also about the state of the environment.

Different data structures have been developed to store and to analyze the information. Every time the user performs an action, all details about those activities are recorded such as what action, with which actor or object, and how many times the user performed this action. Other data is also collected. This includes the state of the user, (health, armor, wealth), the environment (civilizations built, quests completed), and the state of the agents in the environment.

Another important component of the knowledge base is functions for maintaining the data structures. The functions can insert the necessary data into the knowledge base. These functions can also be able to retrieve all the details about the last *n* activities the user has performed (the number of activities can be changed with a few simple code modifications). These facts are updated every time the details of that activity are inserted into the data structures. Some other functions can retrieve the most frequently performed actions, and the object or agent that was interacted with most frequently.

Rule Base

The rule base is the brain of the entire project. This determines the intention of the user. The rule base uses data from the knowledge base and world states to infer users' goals and intentions. Based on this information, the probability of individual user's goals will either increase or decrease. This change might be gradual or sudden based on the actions user performed. For example, hiring

samurai soldiers may increase the probability of pursuing the KNIGHTHOOD goal, but killing a priest may reduce the probability of pursuing KNIGHTHOOD.

The rule base has sets of rules that reason about the situation at different levels of user interactions in the environment. This module will get the basic information from the sensors and from the world state module. The information that the rule base module contains will change the information in the history and user model state modules and those changes again change the information at the rule base module. The relation with the above two modules is co-dependent. Two levels of rules are developed in the rule base to deal with different modules appropriately.

Low-level rules: examine the world state and sensor information and determine if a particular event or interaction should be recorded in the history. For example:

- If user approaches an agent (sensor) then record USER-APPROACHES-AGENT in history
- If user interacts with an agent (sensor) then record USER-INTERACTS-AGENT in history

Mid-level rules: work with the history, world state and sensors to make inferences regarding the users' near-term objectives. For example:

- If user has approached agent (history) and user is still near agent after 10 seconds (world state) then record USER-INTERESTED-IN-AGENT in user model state

- If USER-INTERESTED-IN-AGENT (user model) and agent has the attribute of HELP-WTH-FIGHT (world state) then record USER-INTERESTED-IN-FIGHTING in user model state

The rule base updates goal values representing the probability that a user is pursuing a given goal. These values are updated based on the facts available from the rule base at any particular instant. Goal values are determined by reviewing the user model, history, world state and sensors. For example:

If USER-INTERESTED-IN-FIGHTING (user model) and USER-APPROACHES-AGENT (history) and agent has attribute FIGHT (world state) then add 5% to goal FIGHT-FOR-KINGDOM, add 5% to goal FIGHT-FOR-REVENGE and subtract 5% from goal SETTLE-PEACEFULLY

CHAPTER 3

IMPLEMENTATION

The user modeling architecture and the maps, models and agents used in the test environment all were created to run in a 3D virtual world. Development for 3D environments differs from traditional programming in some respects. I used 3D Game Studio (3DGS) ® (Conitec Datensysteme GmbH, Germany, http://www.conitec.com) for this purpose. In addition to writing programs or scripts to control events and behaviors in the environment, the developer of a 3D environment must also assemble and integrate a number of additional resources. These resources included 3D models, graphic skins for the models, map entities, sprites, prefabs and other 3D entities. There was also a map, or terrain, with physical landscape and the placement of models within the world. The development and assembly of these resources are described in the following sections.

Environment Development

Many tools exist for developing 3D virtual environments, from research-oriented VR applications such as WorldUp® from SENSE8 or the CAVELib™ to more entertainment oriented 3D game engines such as Quake™ (Copyright © 2003 GameSpy Industries, Inc.), or Unreal Tournament (©1999 - 2001 Epic Games Inc.).

The 3D engine used in the development of the test environment is the 3DGS. 3DGS is the leading authoring system for 2D and 3D computer games. It combines a programming language with a high-end 3D engine, a 2D engine, a map and model editor and huge libraries of 3D objects, textures and prefabs. The software is an authoring system used for building 2D and 3D single and multi-player computer games and virtual world interfaces. It combines a 3D engine (A5), a map editor (WED), a model editor (MED), a compiler, libraries of pre-made 3D objects (prefabs) and model maps. It also includes its own scripting language for detecting events and pre defined object behaviors. The 3D game engine is the core of the development system. The engine manages the 3D views and controls the behavior and actions of the user and actors. The A5 engine handles indoor and outdoor sceneries. It also has a lighting engine that supports true shadows and moving light sources.

The scripting language is called C-Script or the World Definition Language. The language is a simplified version of the high-level language C++, adapted to suit the game programming needs using the 3D engine. It supports multitasking, arrays, structs, pointers, string and vector functions, file I/O, etc similar to most other high level programming languages. The software also includes a DLL interface, which allows users to add new effects and features as built-ins. The C-Script is developed to be a compiled language.

The level, model and terrain editors are used to create landscapes, place various effects, and define movement paths and model actors. A library of

prefabricated textures, building parts, furniture, vehicles, weapons and actors is included with the software. Animated 3D models are created with the model editor MED. The model editor enables the user to develop 3D models, built either from scratch using the basic shapes provided, or by import from other model generating tools like Lightwave and Milkshape. Using the world (map) and model editors the different levels of a game or environments are created. The virtual environment is created by generating a terrain map using the world editor, placing different prefabricated objects in it. The models created by the model editor are placed in this environment and are made to follow certain predefined or user-defined actions. These actions are essentially, compiled C-Script functions and code. A number of such predefined functions are included along with the software to provide default actions etc. to characters in the environment.

Model Building

Most model editor programs, including MED, are able to import different 3D formats, like 3DS, X, or MD2, and convert them into the MDL format used by the 3DGS. The particular models for the test bed environment were chosen based upon the needs of the story theme. The models are described in the following section.

Models

Models are entities that exist in the virtual environment. A model is an animated 3-Dimensional shape depicting a real life entity. In the 3D graphical environment, a model is generated as a mesh constituting triangles and polygons

to define the shape of the model. Model entities are used for animated objects like actors and characters in the virtual environment. They are created using a model editor program, like MED, the model editor that comes with 3DGS.

Queen model: This is the model used for the main character. Queen model is assigned the behavior of the user. This model acts as the user and in other words this model represents the user in virtual environment. Model is shown in fig (3).

Knight model: This model is also used for the user. The knight model replaces the queen model that represents the user when the user becomes a knight. This model is shown in fig (4).

Samurai model: This model is used to represent the samurai character. This is a pre-developed model. These models are placed randomly in the environment and new models are generated as the story progresses. This model is shown in fig (5).

Monster model: This model is used to show the monster in the environment. This is a fully developed model with nice textures and colors. This model has many animations for different purposes. These models are placed randomly in the environment and new models are generated as the story progresses. This model follows the user. This model is shown in fig (6).

Scorpion monster model: This model is used to show the big monster. Combining two different models monster and scorpion models makes this model. To get this model the scorpion model is combined with the monster model. The new model will have the textures of the parent model. In this case the parent model is the monster model. This model is shown in the fig (7).

Villager model: This model is also a pre-developed model, used to represent the villager character in the environment. These models are placed randomly in the environment at the village and new models are generated as the story progresses in side the village. This model is shown in fig (8).

Priest model: This model is used to represent the priest character in the environment. Two priests are located inside the church and are shown in fig (9).

Enemy model: This model is a combination of a dragon and warrior models. This model will show the walking animation in the environment, but it also has other animations. This model is shown in fig (10).

Health pack model: This model does not have any animations and is shown in the fig (11).

Armor pack model: This model is made by combining many weapon models to represent a single model. This model is shown in fig (12).

Fig (3)

Fig (4)

Fig(5)

Fig (6)

Fig (7)

Fig (8)

Fig (9)

Fig (10)

Fig (11)

Fig (12)

Skins

A model consists of a continuous 3D mesh with a soft, stretching “skin” wrapped around it. The skin is a picture or a texture that is wrapped around a model. This picture is created using a painting or image editing software, imported into the model editor, which maps the skin on to the 3D mesh.

Animation

“Animation is the ‘liveliness’ of the model that gives it the appearance of movement.” [6] Animations are created by manipulating elements of a model over a series of frames, which are named and numbered according to the action. For example, the frames for the walk animation are named walk1, walk2 and so on. Cycling through the different frames at a specified rate creates the illusion of a motion, which is called animation. The animation is carried out by means of a script, which uses the starting and the ending frame number for that particular animation.

Map Building

The map refers to a 3D space (or environment) in the story in which different characters (or models) interact with each other. In other words it is the virtual environment without any actors or active objects. The map is developed according to the story line and theme. The map is built using the World Editor (WED) provided by 3DGS. The editor provides various options and resources to build the environment.

The map is built from 3D objects that consist of basic blocks, image patterns called textures, mapped onto the surfaces of the blocks. The blocks can be of any shape, and their material can have certain properties, like being passable or impassable, transparent, opaque or even invisible.

The WED editor saves and loads maps and prefabricated 3D objects as WMP files. The texture manager stored the textures that are generated using a paint or image-editing program in WAD collections. Within a map, further maps can be placed, as well as other objects like models, sprites or terrains. The engine is the “core” program, which runs the game and displays the 3-D world on the screen, needs the level in a final WMB format. This format contains some pre-calculated data and is compiled from the WMP file and one or more WAD texture resources once the map is built.

Terrain

A terrain is a large outdoor area bounded by a horizon. Most maps built in WED consist of a terrain. WED inserts a default terrain on to the map. The map builder can choose the terrain to be different than the default one provided by the editor, by specifying the terrain file name in the initializing script used by the engine to load the map. A terrain consists of texture mapped onto a rectangular grid of height values. The terrain file is essentially a 2D image file, which is converted into a depth-mapped 3D object based on the differences in the intensity of color in the 2D image. This conversion is done using the MED (model editor) which converts a bitmap (.bmp) image into a height map (.hmp) terrain object.

The map for the test bed environment was built using basic blocks like large flat plates, cubes etc. The huge floor of the map is mapped with green grass texture, with invisible and impassable walls to prevent the user to fell down accidentally. A large number of tree models, bushes and tree groups are placed in the terrain to give the look of a forestland. A monster place with skeletons, blood and dead bodies also included, giving horror feeling of the monster. Few terrain files including hills and mountains are also added to give proper look and feel for the environment. Prefabs are also added to the environment and are explained below.

Prefabs

After the initial ‘ground’ map is built, the level is populated with a number of models and ‘prefabs’, which give life to the otherwise barren map. A prefab is a prefabricated map or model, simple or complex depending on its configuration and purpose, built in the WED or MED using simple basic shapes. The prefabs I used in the environment includes the forest models made by combining various tree models and shrub models, enemy fort wall and gate, a building with different levels, a church and a village prefab that contains many houses and other prefabs. A prefab may be made to move as a whole in the environment, but it cannot be animated.

Models

Models or entities are things, which can move from one place to another in the environment, animate and can react to different circumstances, or to each

other depending on the behavior attached to them. A model is assigned more or less intelligent behavior. This behavior can be chosen from a set of predefined actions in the WED or created through scripts to incorporate specific functionalities. Models are placed in the environment by loading the appropriate file and positioning them in the environment. The models can also be located dynamically in the environment with the code written appropriately. A model can also be assigned as invisible, transparent, bright etc., depending on its function in the environment.

Event-Driven Programming

3DGS environment, like many 3D environments, can be described as an event driven programming environment. This means that the system is set up to respond to certain circumstances or user interactions (events) by triggering corresponding behaviors (actions and functions).

Behaviors in the 3DGS are controlled through functions and actions. Actions are code modules (like a function) directly associated with a character and typically update attributes of the particular character, such as movements and animations. Functions are also supported by 3DGS. Functions in 3DGS are more general than actions and can update any aspect of the environment, from character movements and animations, to global variables and system flags.

Actions

aQueen: This action is assigned to the user and is the primary control of the user character. This action is responsible for triggering all the events associated to the

user with other actors and objects. These events include killing the actors, hiring or asking help from other actors, stay close to other models and objects and picking up objects like medical pack and armor packs.

aMonster & aEnemy: These actions are the primary controls for the behavior of the monster and enemy characters respectively. These actions are responsible for triggering the events associated with monster and enemy characters. These events include following the user and falling when killed.

aSamurai, aVillager, aPriest, aHealth_pac & aArmor_pac : These actions are the primary control for the behavior of respective models. These actions allow the models to be scanned by the user and to trigger appropriate event based on the user interaction.

User Modeling Functions

Functions are used to describe general system capabilities. These include various operations such as maintaining the knowledge base and maintaining data related to actors, objects and states in the virtual environment. It also includes functions used for user modeling, which are described in brief below:

function print(string)

This function is used to print a given string on the console.

function help_me()

function kill_me()

function pick_me()

These functions are used to reset the action variables.

function game_info()

This function is used to check the world status and inform necessary information to the user when certain conditions are satisfied.

function calculate_Time()

This function is used to calculate the relative time from the starting of the program.

function vec_randomize(&vec,range)

function part_alphafade()

function effect_explo()

These functions are used to produce particle effects when user picked the armor pack or health pack.

function roll_data()

This function is used to display data on the console for user's information. The text will be rolled from the bottom.

function show_details()

This function is used to display the user's attributes on the console.

function built_town()

This function is used to build a new town or civilization at a particular location when user has certain attributes.

function knight()

This function is used to change the queen model of the user to a knight model.

function Queenscan()

This one of the important function that checks for all interactions user has with other objects and actors.

function monster_scan()

This function used to check for monsters behavior when user approaches the monster.

function create_mon()

This function is used to create monsters at various locations in the environment at the beginning of the game. It also decides the locations of the monsters that are created in the middle of the game.

function smonster_scan()

This function used to check for scorpion monsters behavior when user approaches it.

function create_smonster()

This function is used to create the scorpion monster.

function create_vil()

This function is used to create villagers at various locations of the village in the environment at the beginning of the game. It also decides the locations of the villagers that are created in the middle of the game.

function create_sam()

This function is used to create samurai soldiers at various locations in the environment at the beginning of the game. It also decides the locations of the samurais that are created in the middle of the game.

function create_health()

This function is used to create health packs at various locations in the environment at the beginning of the game.

function create_armor()

This function is used to create armor packs at various locations in the environment at the beginning of the game.

function enemy_scan()

This function used to check for enemy soldiers behavior when user approaches them.

function create_ene()

This function is used to create monsters at various locations inside the enemy fort in the environment. It also decides the new locations of the enemy soldiers that flew away from the fort.

function create_pre()

This function is used to create priests at various locations of the inside the church in the environment at the beginning of the game. It also decides the locations of the priests that are created in the middle of the game.

function insert(model_num, action_num, combo, time_var)

This function is used to insert the details of the actions of the user inside a data structure. This will insert four values each time it has been called.

function insertshow()

This function is used to display all the inserted data into the data structure.

function order_arrange()

This function is used to arrange the inserted data in the order of the recent action details to the earliest action details in another data structure.

function reset_MODEL()

function reset_ACTION()

function reset_Combo()

These functions are used to reset certain data structure variables.

function find_maxModel()

This function is used to find out the model with which the user interacted more in the last 10 actions.

function find_maxAction()

This function is used to find out the action, which the user performed more in the last 10 actions.

function find_maxCombo()

This function is used to find out a specific action with a specific actor or object that user performed more in the last 10 actions.

function rulebase_recent()

This function is used to analyze the users intentions based on the actions that the user performed recently.

function rulebase_history()

This function is used to analyze the users intentions based on the past history of events

CHAPTER 4

RESULTS

User modeling is a complex endeavor, and determining user goals by observing events in a rich environment is challenging. The results with successfully predicting users' intentions given the current rule-base have been largely successful. The rulebase has been able to accurately infer users' intention regarding most of the goals under most circumstances. However, a number of issues have affected the overall accuracy. Certain sequences of actions have confused the rule-base, resulting in in-accurate inferences. For example, if a user alternates between actions that support two different goals, the system is unable to accurate infer any single goal, and the results are poor. Also, the WARRIOR goal is sometime over-represented, due to the fact that the actions that support this goal are shared by many other goals. Still, I feel that the project has succeeded in demonstrating that sensors, a knowledge base that includes an event history and a rule-based inference mechanism can be successfully used to determine user goals in a virtual environment.

CHAPTER 5

CONCLUSION

Virtual environments are growing increasingly complex and the designers of virtual environments require increasingly sophisticated tools for creating dynamic experiences that can adapt to user needs. I have demonstrated an architecture for user modeling in virtual environments. Such an architecture could be used in conjunction with virtual environments devoted to a variety of applications, including education, visualization, and entertainment. I feel that architectures such as this will allow the creation of more engaging, more immersive, and more effective experiences in virtual environments.

APPENDIX

SOURCE CODE

The source code is written .wdl files and contains the following files.

Final.wdl, Auxfun.wdl, Queen.wdl, Monster.wdl, Villager.wdl, Samurai.wdl, Enemy.wdl, Healthpac.wdl, Armorpac.wdl, Priest.wdl, Datastruct.wdl, Rulebase.wdl

Final.wdl

```
////////////////////////////////////////////////////////////////////////////
// A5 main wdl
////////////////////////////////////////////////////////////////////////////
// Files to over-ride:
// * logodark.bmp - the engine logo, include your game title
// * horizon.pcx - A horizon map displayed over the sky and cloud maps
////////////////////////////////////////////////////////////////////////////
// The PATH keyword gives directories where game files can be found,
// relative to the level directory
path "D:\\Program Files\\GStudio\\template";  // Path to WDL templates subdirectory

////////////////////////////////////////////////////////////////////////////
// The INCLUDE keyword can be used to include further WDL files,
// like those in the TEMPLATE subdirectory, with prefabricated actions
include <movement.wdl>;
include <messages.wdl>;
include <menu.wdl>;   // must be inserted before doors and weapons
include <particle.wdl>; // remove when you need no particles
include <doors.wdl>;   // remove when you need no doors
include <actors.wdl>;   // remove when you need no actors
include <weapons.wdl>;  // remove when you need no weapons
include <war.wdl>;      // remove when you need no fighting
//include <venture.wdl>; // include when doing an adventure
include <lflare.wdl>;   // remove when you need no lens flares

include <auxfun.wdl>;
include <Queen.wdl>;
include <Monster.wdl>;
include <villager.wdl>;
include <Samurai.wdl>;
include <healthpac.wdl>;
include <armorpac.wdl>;
include <enemy.wdl>;
include <priest.wdl>;
include <datastruct.wdl>;
include <rulebase.wdl>;

////////////////////////////////////////////////////////////////////////////
// The engine starts in the resolution given by the follwing vars.
var video_mode = 6;  // screen size 640x480
var video_depth = 16; // 16 bit colour d3d mode

////////////////////////////////////////////////////////////////
// Strings and filenames
// change this string to your own starting mission message.
string mission_str = "Fight your way through the level. Press [F1] for help";
string level_str = <final.WMB>; // give file names in angular brackets

////////////////////////////////////////////////////////////////
// define a splash screen with the required A4/A5 logo
bmap splashmap = <logodark.pcx>; // the default logo in templates
panel splashscreen {
  bmap = splashmap;
```

```
  flags = refresh,d3d;
}

////////////////////////////////////////////////////////////////////////////
// The following script controls the sky
sky horizon_sky {
  // A backdrop texture's horizontal size must be a power of 2;
  // the vertical size does not matter
  type = <horizon.pcx>;
  tilt = -10;
  flags = scene,overlay,visible;
  layer = 3;
}

////////////////////////////////////////////////////////////////
// The main() function is started at game start
function main()
{
  // set some common flags and variables
  // warn_level = 2; // announce bad texture sizes and bad wdl code
  tex_share = on;// map entities share their textures

  // center the splash screen for non-640x480 resolutions, and display it
  splashscreen.pos_x = (screen_size.x - bmap_width(splashmap))/2;
  splashscreen.pos_y = (screen_size.y - bmap_height(splashmap))/2;
  splashscreen.visible = on;
  // wait 3 frames (for triple buffering) until it is flipped to the foreground
  wait(3);

  // now load the level
  level_load(level_str);
  // freeze the game
  freeze_mode = 1;

  // wait the required second, then switch the splashscreen off.
  sleep(1);
  splashscreen.visible = off;
  bmap_purge(splashmap);// remove splashscreen from video memory

  // load some global variables, like sound volume
  load_status();

  // display the initial message
  msg_show(mission_str,10);

  // initialize lens flares when edition supports flares
  ifdef CAPS_FLARE;
  lensflare_start();
  endif;

  // use the new 3rd person camera
  move_view_cap = 1;

  // un-freeze the game
  freeze_mode = 0;

  // client_move();  // for a possible multiplayer game
  // call further functions here...

  randomize();
  create_monster();
  create_villager();
  create_samurai();
  create_healthpac();
  create_armorpac();
```

```
  create_priest();
  startingtime = (sys_hours * 3600) + (sys_minutes * 60) + (sys_seconds);
  while(1) {
    game_info();
    waitt(16);
  }
}

////////////////////////////////////////////////////////////////
// The following definitions are for the pro edition window composer
// to define the start and exit window of the application.
WINDOW WINSTART
{
  TITLE       "3D GameStudio";
  SIZE      480,320;
  MODE      IMAGE; //STANDARD;
  BG_COLOR     RGB(240,240,240);
  FRAME       FTYP1,0,0,480,320;
  // BUTTON     BUTTON_START,SYS_DEFAULT,"Start",400,288,72,24;
  BUTTON     BUTTON_QUIT,SYS_DEFAULT,"Abort",400,288,72,24;
  TEXT_STDOUT "Arial",RGB(0,0,0),10,10,460,280;
}

/* no exit window at all..
WINDOW WINEND
{
  TITLE       "Finished";
  SIZE      540,320;
  MODE      STANDARD;
  BG_COLOR     RGB(0,0,0);
  TEXT_STDOUT  "",RGB(255,40,40),10,20,520,270;

  SET FONT     "",RGB(0,255,255);
  TEXT        "Any key to exit",10,270;
}*/


////////////////////////////////////////////////////////////////
//INCLUDE <debug.wdl>;
```

Auxfun.wdl

```
var startingtime;

function create_smonster(); // function declaration. this function is defined in monster.mdl

//variables to give numbers to MODELS
var model_name = 0;
var model_monster = 1;
var model_samurai = 2;
var model_villager = 3;
var model_priest = 4;
var model_enemy = 5;
var model_healthpac = 6;
var model_armorpac = 7;

//variables to give numbers to ACTIONS
var action_name = 0;
var action_kill = 1;
var action_hire = 2;
var action_pick = 3;
var action_stay = 4;
```

```
//variables given to combination of models & actions

var combo_name = 0;
var combo_moki = 1;
var combo_saki = 2;
var combo_sahi = 3;
var combo_sast = 4;
var combo_viki = 5;
var combo_vihi = 6;
var combo_vist = 7;
var combo_prki = 8;
var combo_prhi = 9;
var combo_prst = 10;
var combo_enki = 11;
var combo_hepi = 12;
var combo_hest = 13;
var combo_arpi = 14;
var combo_arst = 15;

var time_spent = 0;

//variables to claculate the # of killed entities
var mon_killed = 0;
var vil_killed = 0;
var sam_killed = 0;
var pre_killed = 0;
var ene_killed = 0;

//variables to claculate the # of user attempts to kill entities
var mon_killtry = 0;
var vil_killtry = 0;
var sam_killtry = 0;
var pre_killtry = 0;
var ene_killtry = 0;

//variables to keep track of hired entities
var vil_hired = 0;
var sam_hired = 0;

//variables to keep track of user tried to get help or hire
var vil_helptry = 0;
var sam_helptry = 0;
var pre_helptry = 0;

//variables to claculate the # of times user approached the entities
var mon_approached = 0;
var vil_approached = 0;
var sam_approached = 0;
var pre_approached = 0;
var ene_approached = 0;
var hea_approached = 0;
var arm_approached = 0;

//variables to calculate # of times user picked the entity pacs
var hea_picked = 0;
var arm_picked = 0;

//variables about the knowledge of world status
var town_knowledge = 0;
var knight_knowledge = 0;
var enemy_knowledge = 0;
var dragon_knowledge = 0;


// variables to find the time, when the knowledge and state changes occur
```

```
var townknow_time = 0;
var knightknow_time = 0;
var enemyknow_time = 0;
var smonsterknow_time = 0;
var townbuilt_time = 0;
var knighthood_time = 0;
var fortcaptured_time = 0;
var smonsterdead_time = 0;

//variables to know about the new states and newly created entities

var town_built = 0;
var knighthood = 0;
var fort_captured = 0;
var smonster_dead = 0;

//variable to check whether user is near to an entity
var away_flag = 1;

//variables for initialising user attributes
var baby_armor = 1050;
var baby_health = 1075;
var baby_power = 1010;
var baby_gold = 1025;

var model_exist = 1;

var helpvar = 0;
var killvar = 0;
var pickvar = 0;

var sam_hfailed = 0;
var sam_hftime = 0;
var vil_hfailed = 0;
var vil_hftime = 0;

var mtemp[3];
var babypos[3];

var count_time;
var present_time;
var diff_time;
var temp_time = 0;
var stringcount = 0;


string ds1 = " Gold";
string ds2 = " Armor";
string ds3 = " Power";
string ds4 = " Health";

string dstemp[50];
string dstemp1[50];
string dstemp2[50];
string dstemp3[50];
string dstemp4[50];

entity* temp_entity;
entity* temp_entity1;
entity* deadmodel;

font basic = <msgfont.pcx>,12,16;
text temp_text {//temporary storage of story text
  pos_x = 5;
  pos_y = 5;
```

```
  font = basic;
  strings = 8;
  layer 2;
}
text my_text { //display for story text
  pos_x = 5;
  pos_y = 5;
  font = basic;
  strings = 8;
  layer 2;
}

// Panel Definition
bmap p1 = <panel4.bmp>;
panel panel_1 {
  bmap = p1;
  layer 1;
  pos_x = 1; pos_y =325;
  flags = refresh,d3d,transparent;
}

function print(str)
{
  my_text.string[0] = str;
  my_text.visible = on;
}

function help_me()
{
  helpvar  = 1;
  WAITT(40);
  helpvar = 0;
}

function kill_me()
{
  killvar  = 1;
  WAITT(40);
  killvar = 0;
}

function pick_me()
{
  pickvar  = 1;
  WAITT(40);
  pickvar = 0;
}

function game_info()
{
  //KNIGHT INFORMATION
  if(knight_knowledge == 0 && sam_hired == 3)  {
    knight_knowledge = 1;
    my_text.string[0] = "If you hire more soldiers and have power and health";
    my_text.string[1] = "Priest will make you KNIGHT by asking him";
    panel_1.visible = on;
    roll_data();
    calculate_time();
    knightknow_time = count_time;
    waitt(320);
    my_text.visible = off;
    waitt(16);
    panel_1.visible = off;
  }
```

```
//ENEMY INFORMATION and DRAGON MONSTER INFORMATION
if(enemy_knowledge == 0 && knighthood)  {
  enemy_knowledge = 1;
  create_enemy();
  my_text.string[0] = "If you hire more soldiers";
  my_text.string[1] = "You can go to attack your enemies";
  calculate_time();
  if(dragon_knowledge == 0) {
    dragon_knowledge = 1;
    my_text.string[2] = "-----------------------------------";
    my_text.string[3] = "If you hire more soldiers";
    my_text.string[4] = "You can kill the '-dragon monster-'";
    smonsterknow_time = count_time;
    create_smonster();
  }
  panel_1.visible = on;
  roll_data();
  enemyknow_time = count_time;
  waitt(320);
  my_text.visible = off;
  waitt(16);
  panel_1.visible = off;
}
else  {
  if(enemy_knowledge == 0 && sam_hired == 5 )  {
    enemy_knowledge = 1;
    create_enemy();
    my_text.string[0] = "If you become a knight and hire more soldiers";
    my_text.string[1] = "You can go to attack your enemies";
    calculate_time();
    if(dragon_knowledge == 0) {
      dragon_knowledge = 1;
      my_text.string[2] = "-----------------------------------";
      my_text.string[3] = "If you become a knight and hire more soldiers";
      my_text.string[4] = "You can kill the '-dragon monster-'";
      smonsterknow_time = count_time;
      create_smonster();
    }
    panel_1.visible = on;
    roll_data();
    enemyknow_time = count_time;
    waitt(320);
    my_text.visible = off;
    waitt(16);
    panel_1.visible = off;
  }
}

if(dragon_knowledge == 0 && mon_killed == 5 ) {
  dragon_knowledge = 1;
  my_text.string[0] = "If you become a knight and kill monsters";
  my_text.string[1] = "You can kill the '-dragon monster-'";
  panel_1.visible = on;
  roll_data();
  calculate_time();
  smonsterknow_time = count_time;
  create_smonster();
  waitt(320);
  my_text.visible = off;
  waitt(16);
  panel_1.visible = off;
}

//TOWN INFORMATION
if(town_knowledge == 0 && ((vil_hired == 3 && baby_gold > 75) || (vil_hired == 6))) {
```

```
    town_knowledge = 1;
    my_text.string[0] = "If you have 8 villagers 100 gold and 25 power";
    my_text.string[1] = "You can build a new TOWN by pressing '-B-'";
    panel_1.visible = on;
    roll_data();
    calculate_time();
    townknow_time = count_time;
    waitt(320);
    my_text.visible = off;
    waitt(16);
    panel_1.visible = off;
  }
}

function calculate_Time()
{
  count_time = ((sys_hours * 3600) + (sys_minutes * 60) + (sys_seconds)) - startingtime;
}

//*** move random  distance & direction, for expolosion effect
function vec_randomize(&vec,range)
{
  vec[0] = random(1) - 0.5;
  vec[1] = random(1) - 0.5;
  vec[2] = random(1) - 0.5;
  vec_normalize(vec, random(range));
}

//*** fade particles
function part_alphafade()
{
  my.alpha -= time+time;
  if (my.alpha <= 0) { my.lifespan = 0; }
}

//*** explosion
function effect_explo()
{
  vec_randomize(temp,15);
  vec_add(my.vel_x, temp);
  my.alpha = 50 + random(25);
  my.bmap = scatter_map;
  my.flare = on;
  my.bright = on;
  my.beam = on;
  my.move = on;
  my.function = part_alphafade;
}

function roll_data()
{
  my_text.pos_y = 450; // move the text below the screen
  my_text.visible = on; // make it visible
  while (my_text.pos_y > 330) { // as long as text still is on-screen
    my_text.pos_y -= 1; // roll upwards one line of pixels
    waitt(1); // then wait for one frame
  }
  waitt(320); // time enough to read the last line

  while(stringcount<8) {
    my_text.string[stringcount] = "";
    stringcount += 1;
  }
  stringcount = 0;
  waitt(16);
```

```
}

function show_details()
{
  STR_FOR_NUM(dstemp1,baby_health);
  STR_CAT(dstemp1,ds4);
  my_text.string[0]= dstemp1;

  STR_FOR_NUM(dstemp2,baby_power);
  STR_CAT(dstemp2,ds3);
  my_text.string[1]= dstemp2;

  STR_FOR_NUM(dstemp3,baby_armor);
  STR_CAT(dstemp3,ds2);
  my_text.string[2]= dstemp3;

  STR_FOR_NUM(dstemp4,baby_gold);
  STR_CAT(dstemp4,ds1);
  my_text.string[3]= dstemp4;

  roll_data();
  waitt(160);
  my_text.visible = off;
  waitt(16);
  panel_1.visible = off;
}

string town1_bmp = <village.bmp>;
string town2_bmp = <village1.bmp>;
string trees1 = <tree01.pcx>;
string trees2 = <tree02.pcx>;
string trees3 = <tree03.pcx>;
string trees4 = <tree04.pcx>;

action adeadmodel
{
  deadmodel = me;
}

var townloc[3];
function built_town()
{
  if(town_knowledge && baby_gold >= 100 && baby_power >= 25 && vil_hired >= 5 ){
    townloc[0] = 500;
    townloc[1] = -8500;
    townloc[2] = 60;
    ent_create(town1_bmp, townloc,adeadmodel);
    townloc[0] = 1500;
    townloc[1] = -8000;
    ent_create(town2_bmp, townloc,adeadmodel);
    townloc[0] = 0;
    ent_create(trees1, townloc,adeadmodel);
    townloc[0] = 2000;
    ent_create(trees2, townloc,adeadmodel);
    townloc[1] = -7500;
    ent_create(trees3, townloc,adeadmodel);
    townloc[0] = 0;
    ent_create(trees4, townloc,adeadmodel);
  }
  calculate_time();
  townbuilt_time = count_time;
  town_built = 1;
}

on_b = built_town();
```

```
on_h = help_me;
on_k = kill_me();
on_p = pick_me();
on_s = show_details();
```

Queen.wdl

```
string deadsamurai_mdl = <deadsamurai.mdl>;
entity* baby;
entity* temp_entity;
function insert(model_num, action_num, combo, time_var);

action babe
{
  baby = me;ptr_for_name("cbabe_mdl_001");
  my._signal = 0;            // player not yet detected
  my.enable_scan = on;       // sensible of explosions
  my.event = Queenscan();
  player_move();
  while(1) {
    babypos[0] = my.x;//baby.x;
    babypos[1] = my.y;//baby.y;
    babypos[2] = my.z;//baby.z;
    waitt(1);
  }
}

entity* test_entity;
string knight_mdl = <knight.mdl>;
var testvec[3];

function knight()
{
  test_entity = baby;
  testvec[0] = 0;
  testvec[1] = 0;
  testvec[2] = -500;
  ent_remove(baby);
  baby = ent_create(knight_mdl,babypos,babe);
  print("Knight developed");
  calculate_time();
  knighthood_time = count_time;
}

var temp1_time = 0;
var temp2_time = 0;
var diff1_time = 0;

function Queenscan()
{
  vec_to_angle(my_angle,temp);
  force = 4;          // set rotation speed
  actor_turn();       // rotate towards my_angle
  LABEL:
  while (1){
    // scan for the player
    temp.pan = 180;
    temp.tilt = 180;
    temp.z = 100;
    indicator = _watch;
    scan(baby.x, baby.pan,temp);
    trace_mode = IGNORE_ME + IGNORE_PASSABLE + IGNORE_PASSENTS;
    //trace(temp_entity.POS,babypos);//temp_entity.POS
```

```
if(RESULT >100 || RESULT <= 0 )   {
  away_flag = 1;
  calculate_Time();
  temp1_time = count_time;
}

if((RESULT > 0) && (RESULT < 100))
{
  //starting of checking for SAMURAI

  if(you.skill3)     {
    MODEL_NAME = MODEL_SAMURAI;
    if(away_flag)
    {
      sam_approached += 1;
      away_flag = 0;
    }
    calculate_Time();
    temp2_time = count_time;
    diff1_time = temp2_time - temp1_time;
    if(diff1_time == 10)      {
      print("User stayed at Samurai");
      waitt(40);
      my_text.visible = off;
      ACTION_NAME = ACTION_STAY;
      calculate_Time();
      TIME_SPENT = count_time;//int(count_time /60);
      insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
      TIME_SPENT = 0;
    }
    temp_entity = ptr_for_handle(you.skill30);
    if(killvar)        {
      ACTION_NAME = ACTION_KILL;
      COMBO_NAME = COMBO_SAKI;
      calculate_Time();
      TIME_SPENT = count_time;//int(count_time /60);
      insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
      if(baby_armor <=3 || baby_power <= 3) {
        print("You dont have enough energy");
        WAITT(40);
        my_text.visible = off;
        sam_killtry += 1;
        print("User tried to kill Samurai, but failed");
        WAITT(40);
        my_text.visible = off;
      }
      else    {
        killvar = 0;
        ENT_REMOVE(temp_entity);
        ENT_CREATE (deadsamurai_mdl, babypos, adeadmodel);
        //create_deadsam();
        create_sam();
        sam_killed += 1;
        str_for_num(dstemp,sam_killed);
        str_cat(dstemp,"  Samurais killed");
        print(dstemp);
        WAITT(40);
        my_text.visible = off;

        sam_killtry += 1;
        baby_gold += 5;
        baby_armor -= 3;
        baby_power -= 3;
        goto LABEL;
```

```
    }
  }
  if(helpvar)   {
    ACTION_NAME = ACTION_HIRE;
    COMBO_NAME = COMBO_SAHI;
    calculate_Time();
    TIME_SPENT = count_time;//int(count_time /60);
    insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
    print("QUEEN : Help me");
    if(baby_gold >= 15 && baby_armor >= 5)    {
      ENT_REMOVE(temp_entity);
      create_sam();
      print("SOLDIER : Sure.... Thanks for the GOLD(15) and ARMOR(5)I will help you");
      WAITT(40);
      my_text.visible = off;
      sam_hired += 1;
      str_for_num(dstemp,sam_hired);
      str_cat(dstemp,"  Samurais hired");
      print(dstemp);
      WAITT(40);
      my_text.visible = off;

      sam_helptry += 1;
      baby_gold -= 15;
      baby_armor -= 5;
      baby_power += 2;
    }
    else  {
      print("SOLDIER : Ooh Queen... I need 15 GOLD and ARMOR to help you");
      WAITT(40);
      my_text.visible = off;
      sam_helptry += 1;
      print("User tried to hire Samurai, but failed");
      WAITT(40);
      my_text.visible = off;

      sam_hfailed = 1;
      sam_hftime = count_time;

    }
    helpvar = 0;
    goto LABEL;
  }
}

//ending of checking for SAMURAI

//starting of checking for VILLAGER
if(you.skill5){
  MODEL_NAME = MODEL_VILLAGER;
  if(away_flag)  {
    vil_approached += 1;
    away_flag = 0;
  }

  calculate_Time();
  temp2_time = count_time;
  diff1_time = temp2_time - temp1_time;
  if(diff1_time == 10)   {
    print("User stayed at Villager");
    waitt(40);
    my_text.visible = off;
    ACTION_NAME = ACTION_STAY;
    COMBO_NAME = COMBO_VIST;
    calculate_Time();
```

```
  TIME_SPENT = count_time;//int(count_time /60);
  insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
  TIME_SPENT = 0;
}
temp_entity = ptr_for_handle(you.skill25);
if(killvar)   {
  ACTION_NAME = ACTION_KILL;
  COMBO_NAME = COMBO_VIKI;
  calculate_Time();
  TIME_SPENT = count_time;//int(count_time /60);
  insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
  if(baby_power < 5)   {
    print("You dont have enough energy");
    WAITT(40);
    my_text.visible = off;
    vil_killtry += 1;
    print("User tried to kill Villager, but failed");
    WAITT(40);
    my_text.visible = off;
  }
  else    {
    killvar = 0;
    ENT_REMOVE(temp_entity);
    vil_killed += 1;
    str_for_num(dstemp,vil_killed);
    str_cat(dstemp,"  Villagers killed");
    print(dstemp);
    WAITT(40);
    my_text.visible = off;
    create_vil();
    vil_killtry += 1;
    baby_power -= 5;
    goto LABEL;
  }
}
if(helpvar)   {
  ACTION_NAME = ACTION_HIRE;
  COMBO_NAME = COMBO_VIHI;
  calculate_Time();
  TIME_SPENT = count_time;//int(count_time /60);
  insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
  print("QUEEN : Help me");
  if(baby_gold >= 15 && baby_power >= 4)    {
    ENT_REMOVE(temp_entity);
    create_vil();
    print("VILLAGER : Sure.... Thanks for the GOLD(15) and you have POWER(4)...I will help you");
    WAITT(40);
    my_text.visible = off;
    vil_hired += 1;
    str_for_num(dstemp,vil_hired);
    str_cat(dstemp,"  Villagers hired");
    print(dstemp);
    WAITT(40);
    my_text.visible = off;

    vil_helptry += 1;
    baby_gold -= 15;
    baby_power -= 4;
    baby_health += 5;
    goto LABEL;
  }
  else    {
    print("VILLAGER : Ooh Queen... I need 15 GOLD and You need to have 4 POWER to help you");
    WAITT(40);
    my_text.visible = off;
```

```
        print("User tried to hire villager, but failed");
        WAITT(40);
        my_text.visible = off;
        vil_helptry += 1;

        vil_hfailed = 1;
        vil_hftime = count_time;
      }
      helpvar = 0;
    }
}
//ending of checking for VILLAGER

//starting of checking for PRIEST

if(you.skill13) {
  MODEL_NAME = MODEL_PRIEST;
  if(away_flag)  {
    pre_approached += 1;
    away_flag = 0;
  }

  calculate_Time();
  temp2_time = count_time;
  diff1_time = temp2_time - temp1_time;
  if(diff1_time == 10)   {
    print("User stayed at Priest");
    waitt(40);
    my_text.visible = off;
    ACTION_NAME = ACTION_STAY;
    COMBO_NAME = COMBO_PRST;
    calculate_Time();
    TIME_SPENT = count_time;//int(count_time /60);
    insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
    TIME_SPENT = 0;
  }
  temp_entity = ptr_for_handle(you.skill39);
  if(killvar)   {
    ACTION_NAME = ACTION_KILL;
    COMBO_NAME = COMBO_PRKI;
    calculate_Time();
    TIME_SPENT = count_time;//int(count_time /60);
    insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
    if(baby_power < 20)    {
      pre_killtry += 1;
      baby_power -= 10;
      print("You dont have enough energy");
      WAITT(40);
      my_text.visible = off;
      print("User tried to kill Priest, but failed");
      WAITT(40);
      my_text.visible = off;
    }
    else    {
      killvar = 0;
      pre_killed += 1;
      pre_killtry += 1;
      baby_power -= 20;
      ENT_REMOVE(temp_entity);
      create_pre();
      str_for_num(dstemp,pre_killed);
      str_cat(dstemp,"  Priests killed");
      print(dstemp);
      WAITT(40);
      my_text.visible = off;
```

```
      goto LABEL;
    }
  }

  if(helpvar)   {
    ACTION_NAME = ACTION_HIRE;
    COMBO_NAME = COMBO_PRHI;
    calculate_Time();
    TIME_SPENT = count_time;//int(count_time /60);
    insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
    pre_helptry += 1;
    print("ASKING FOR PRIEST HELP");
    WAITT(40);
    my_text.visible = off;
    if(knighthood != 1 )   {
      if(baby_power > 30 && baby_health >= 25 && sam_hired >= 5){
        knight();
        knighthood = 1;
        baby_armor += 25;
        baby_power += 15;
        sam_hired -= 5;
        while(1) {
          if(baby_power < 50){
            baby_power += 2;
          }
          if(baby_armor < 50){
            baby_armor += 2;
          }
          waitt(200);
        }
      }
      else  {
        print("If you have power, health and 5 soldiers I can make you knight");
        WAITT(40);
        my_text.visible = off;
      }
    }
    else  {
      print("you achieved the KNIGHTHOOD continue on your quest");
      WAITT(40);
      my_text.visible = off;
    }
    helpvar = 0;
  }
}

//ending of checking for PRIEST

//starting of checking for HEALTHPAC      //if(model_exist)       //{
  if(you.skill7){
    MODEL_NAME = MODEL_HEALTHPAC;
    if(away_flag)  {
      hea_approached += 1;
      away_flag = 0;
    }
    calculate_Time();
    temp2_time = count_time;
    diff1_time = temp2_time - temp1_time;
    if(diff1_time == 10) {
      print("User stayed at Healthpac");
      waitt(40);
      my_text.visible = off;
      ACTION_NAME = ACTION_stay;
      COMBO_NAME = COMBO_HEST;
      calculate_Time();
```

```
    TIME_SPENT = count_time;//int(count_time /60);
    insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
    TIME_SPENT = 0;
  }
  if(pickvar) {
    ACTION_NAME = ACTION_PICK;
    COMBO_NAME = COMBO_HEPI;
    calculate_Time();
    TIME_SPENT = count_time;//int(count_time /60);
    insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
    temp_entity = ptr_for_handle(you.skill21);
    calculate_time();
    present_time = count_time;
    diff_time = present_time - temp_time;

    if(pickvar && diff_time > 10){
      vec_scale(normal,3);   //*** produce bigger explosion
      effect(effect_explo,2000,MY.pos,normal);

      hea_picked += 1;
      baby_health += (5 + int(RANDOM(15)));
      calculate_time();
      temp_time = count_time;
      pickvar = 0;
    }
  }
}
//ending of checking for HEALTHPAC

//starting of checking for ARMORPAC

if(you.skill9){
  MODEL_NAME = MODEL_ARMORPAC;
  if(away_flag)  {
    arm_approached += 1;
    away_flag = 0;
  }
  calculate_Time();
  temp2_time = count_time;
  diff1_time = temp2_time - temp1_time;
  if(diff1_time == 10){
    print("User stayed at Armorpac");
    waitt(40);
    my_text.visible = off;
    ACTION_NAME = action_stay;
    COMBO_NAME = COMBO_ARST;
    calculate_Time();
    TIME_SPENT = count_time;//int(count_time /60);
    insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
    TIME_SPENT = 0;
  }

  if(pickvar) {
    ACTION_NAME = ACTION_PICK;
    COMBO_NAME = COMBO_ARPI;
    calculate_Time();
    TIME_SPENT = count_time;//int(count_time /60);
    insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
    temp_entity = ptr_for_handle(you.skill27);
    calculate_time();
    present_time = count_time;
    diff_time = present_time - temp_time;

    if(pickvar && diff_time > 10)   {
      vec_scale(normal,3);   //*** produce bigger explosion
```

```
            effect(effect_explo,2000,MY.pos,normal);

            arm_picked += 1;
            baby_armor += (5 + int(RANDOM(15)));
            calculate_time();
            temp_time = count_time;
            pickvar = 0;
          }
        }
      }

      //ending of checking for ARMORPAC

    }//end of SCAN/TRACE IF
    waitt(8);
  }//end of WHILE
}
```

Monster.wdl

```
function create_mon();
var m_start_pos[90] = 1000, 500, 60, -1100, 600, 60, 0, -1000, 60, -3750, 6350, 60;
var mindex = 0;
var testvec[3];
var mtemp1[3];

entity* mon_entity;
string monster_mdl = <monster.mdl>;
string deadmonster_mdl = <deadmonster.mdl>;

function insert(model_num, action_num, combo, time_var);

action amonster
{
  my.enable_detect = on; // sensible of the player
  my.event = monster_scan() ;
}

var mon_follow_count = 0;
var smon_follow_count = 0;

function monster_scan()
{
  vec_to_angle(my_angle,temp);
  force = 4;          // set rotation speed
  actor_turn();       // rotate towards my_angle

  while (1) {
    temp_entity = me;
    testvec[0] = 0;
    testvec[1] = 0;
    testvec[2] = -500;

    // scan for the player

    mtemp1.pan = 180;
    mtemp1.tilt = 180;
    mtemp1.z = 100;
    indicator = _watch;
    trace_mode = IGNORE_ME + IGNORE_PASSABLE + IGNORE_PASSENTS;
    trace(my.POS,babypos);

    if((RESULT > 0) && (RESULT < 100) && (YOU == player) )   // spotted the user!
```

```
    {
      actor_follow();
      beep();

      MODEL_NAME = MODEL_MONSTER;
      mon_follow_count += 0.5;
      if(mon_follow_count > 10 && baby_armor > 1 && baby_health > 1)
      {
        baby_armor -= 2;
        baby_health -= 2;
        mon_follow_count = 0;
      }

      if(killvar)
      {
        ACTION_NAME = ACTION_KILL;
        COMBO_NAME = COMBO_MOKI;
        calculate_Time();
        TIME_SPENT = count_time;
        insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);
        if(baby_armor <=10 || baby_health <= 20)
        {
          print("You dont have enough energy");
          WAITT(40);
          my_text.visible = off;
          mon_killtry += 1;
          print("User tried to kill Monster, but failed");
          WAITT(40);
          my_text.visible = off;
        }
        else
        {
          killvar = 0;
          temp_entity.visible = off;
          temp_entity.passable = on;
          ent_move(testvec, nullvector);
          create_mon();
          mon_killed += 1;
          str_for_num(dstemp,mon_killed);
          str_cat(dstemp,"  Monsters killed");
          print(dstemp);
          waitt(40);
          my_text.visible = off;

          mon_killtry += 1;
          baby_gold += 20;
          baby_armor -= 5;
          baby_power += 5;
          baby_health -= 15;
        }

      }
    }
    waitt(8);
  }
}

function create_mon()
{
  if(mindex < 12)
  {
    temp.x = m_start_pos[mindex];
    temp.y = m_start_pos[mindex + 1];
    temp.z = m_start_pos[mindex + 2];
    mindex += 3;
```

```
  }
  else
  {
    temp.x = babypos[0] + RANDOM(500) + RANDOM(700) + RANDOM(1500) + RANDOM(500) + RANDOM(500) +300;
    temp.y = babypos[1] + RANDOM(700) + RANDOM(300) + RANDOM(1000) + RANDOM(500) + 200;
    temp.z =  60;
    if(temp.x > ABS(8000) || temp.y > ABS(8500))
    {
      mindex = 0;
      temp.x = m_start_pos[mindex] + RANDOM(1000);
      temp.y = m_start_pos[mindex + 1] + RANDOM(1000);
      temp.z = m_start_pos[mindex + 2];
      mindex += 3;
    }
  }
  mon_entity = ent_create(monster_mdl,temp,amonster);
  mon_entity.skill1 = 1;
  mon_entity.skill40 = HANDLE(mon_entity);
  print("Monster created");
  waitt(40);
  my_text.visible = off;
}

function create_monster()
{
  create_mon();
  create_mon();
  create_mon();
  create_mon();
}

entity* smon_entity;
string smonster_mdl = <smonster.mdl>;

action asmonster
{
  my.enable_detect = on; // sensible of the player
  my.event = smonster_scan() ;
}

function create_smonster()
{
  temp.x = -4300;
  temp.y = 7350;
  temp.z = 125;

  smon_entity = ent_create(smonster_mdl,temp,asmonster);
  smon_entity.skill15 = 1;
  smon_entity.skill17 = HANDLE(smon_entity);
}

function smonster_scan()
{
  vec_to_angle(my_angle,temp);
  force = 4;          // set rotation speed
  actor_turn();       // rotate towards my_angle

  while (1)
  {
    temp_entity = me;
    testvec[0] = 0;
    testvec[1] = 0;
    testvec[2] = -500;

    // scan for the player
```

```
    mtemp1.pan = 180;
    mtemp1.tilt = 180;
    mtemp1.z = 100;
    indicator = _watch;
    trace_mode = IGNORE_ME + IGNORE_PASSABLE + IGNORE_PASSENTS;
    trace(my.POS,babypos);//temp_entity.POS

    if((RESULT > 0) && (RESULT < 100) && (YOU == player) )   // spotted the user!
    {
      actor_follow();
      beep();
      smon_follow_count += 0.5;
      if(smon_follow_count > 10 && baby_armor > 5 && baby_health > 5)
      {
        baby_armor -= 5;
        baby_health -= 5;
        smon_follow_count = 0;
      }
      if(killvar)
      {
        if(knighthood && baby_armor <= 40 && baby_health <= 40)
        {
          print("You need 40 health and 40 armor");
          WAITT(40);
          my_text.visible = off;
          print("User tried to kill Scorpion Monster, but failed");
          WAITT(40);
          my_text.visible = off;

          baby_armor -= 5;
          baby_health -= 5;
        }
        else
        {
          killvar = 0;
          temp_entity.visible = off;
          temp_entity.passable = on;
          ent_move(testvec, nullvector);
          waitt(40);
          my_text.visible = off;

          calculate_Time();
          smonsterdead_time = count_time;
          smonster_dead = 1;

          baby_gold += 125;
          baby_armor -= 25;
          baby_power += 75;
          baby_health -= 25;
        }

      }
    }
    waitt(8);
  }
}
```

Villager.wdl

```
var v_start_pos[90] =  -3500, -7600, 75, -4400, -8350, 65, -5725, -7423, 92, -5480, -6850, 86, -4300, -6900, 70, -4400, -7700,
70, -4785, -7000, 70;
var vindex = 0;
```

```
entity* vil_entity;
string villager_mdl = <villager.mdl>;

action avillager
{
  my.enable_scan = on;       // sensible of explosions
  my.enable_detect = on; // sensible of the player
}

function create_vil()
{
  if(vindex >= 21)//locating the VILLAGERs at the predefined locations
  {
    vindex = 0;
  }
  temp.x = v_start_pos[vindex] + random(25);
  temp.y = v_start_pos[vindex + 1] + random(15);
  temp.z = v_start_pos[vindex + 2] + random(5);
  vindex += 3;

  vil_entity = ENT_CREATE (villager_mdl, temp, avillager);
  vil_entity.skill5 = 1;
  vil_entity.skill25 = HANDLE(vil_entity);
  print("villager created");
  waitt(40);
  my_text.visible = off;
}

//creating 5 VILLAGERs
function create_villager()
{
  create_vil();
  create_vil();
  create_vil();
  create_vil();
  create_vil();
}
```

Samurai.wdl

```
var s_start_pos[90] = 6300, 1100, 60, 3000, -2000, 60, 1100, -6800, 60, -5000, 0, 60, -5700, 4000, 60;
var sindex = 0;

entity* sam_entity;
string samurai_mdl = <samurai.mdl>;

action asamurai
{
  my.enable_scan = on;       // sensible of explosions
  my.enable_detect = on; // sensible of the player
}

function create_sam()
{
  //placing the first 5 SAMURAIs at predefined locations
  if(sindex < 15)
  {
    temp.x = s_start_pos[sindex];
    temp.y = s_start_pos[sindex + 1];
    temp.z = s_start_pos[sindex + 2];
    sindex += 3;
  }
```

```
  else       //placing new SAMURAIs randomly around USER's position
  {
    temp.x = babypos[0] + RANDOM(500) + RANDOM(700) + RANDOM(1500) + RANDOM(500) + RANDOM(500) +300;
    temp.y = babypos[1] + RANDOM(700) + RANDOM(300) + RANDOM(1000) + RANDOM(500) + 200;
    temp.z =  60;
    if(temp.x > ABS(8000) || temp.y > ABS(8500))  //checking for bounds
    {
      sindex = 0;
      temp.x = s_start_pos[sindex] + RANDOM(1000);
      temp.y = s_start_pos[sindex + 1] + RANDOM(1000);
      temp.z = s_start_pos[sindex + 2];
      sindex += 3;
    }
  }
  sam_entity = ENT_CREATE (samurai_mdl, temp, asamurai);
  sam_entity.skill3 = 1;
  sam_entity.skill30 = HANDLE(sam_entity);
}

//creating 5 initial SAMURAIs
function create_samurai()
{
  create_sam();
  create_sam();
  create_sam();
  create_sam();
  create_sam();
}
```

Enemy.wdl

```
var e_start_pos[90] = 5650,5050,50, 4070,6080,50, 4400,5750,50, 4100,7000,50, 5100,6700,50, 5500,7600,50,
3300,7550,50, 5000,8200,50, 3000,8500,50, 6000,7000,50;
var eindex = 0;
var testvec[3];
var etemp1[3];

var eneescape_time = 999999;
var ene_need_kill = 15;

entity* ene_entity;
string enemy_mdl = <enemy.mdl>;
string deadenemy_mdl = <deadenemy.mdl>;

function insert(model_num, action_num, combo, time_var);

action aenemy
{
  my.enable_detect = on; // sensible of the player
  my.event = enemy_scan() ;
}

var ene_follow_count = 0;

function enemy_scan()
{
  vec_to_angle(my_angle,temp);
  force = 4;          // set rotation speed
  actor_turn();       // rotate towards my_angle

  while (1)
  {
    temp_entity = me;
```

```
//vector to move the dead enemies
testvec[0] = 0;
testvec[1] = 0;
testvec[2] = -500;

// scan for the player

mtemp1.pan = 180;
mtemp1.tilt = 180;
mtemp1.z = 100;
indicator = _watch;
trace_mode = IGNORE_ME + IGNORE_PASSABLE + IGNORE_PASSENTS;
trace(my.POS,babypos);//temp_entity.POS

if((RESULT > 0) && (RESULT < 300) && (YOU == player) )   // spotted the user!
{
  actor_follow();
  beep();

  MODEL_NAME = MODEL_ENEMY;

  //checking the time the ENEMY near to the USER
  ene_follow_count += 0.5; //equals to 0.5 sec
  if(ene_follow_count > 10 && baby_armor > 0 && baby_health > 0)
  {
    baby_armor -= 1;
    baby_health -= 1;
    ene_follow_count = 0;
  }

  if(killvar)
  {
    ACTION_NAME = ACTION_KILL;
    COMBO_NAME = COMBO_ENKI;
    calculate_Time();
    TIME_SPENT = count_time;//int(count_time /60);
    insert(MODEL_NAME,ACTION_NAME,COMBO_NAME,TIME_SPENT);

    if(baby_health <= 10 || (knighthood == 0) || sam_hired < 5)
    {
      print("You needs knighthood, 5 soldiers and 15 health");
      WAITT(40);
      my_text.visible = off;
      ene_killtry += 1;
      print("User tried to kill the enemy, but failed");
      WAITT(40);
      my_text.visible = off;
    }
    else
    {
      killvar = 0;
      temp_entity.visible = off;
      temp_entity.passable = on;
      ent_move(testvec, nullvector);
      ene_killed += 1;
      str_for_num(dstemp,ene_killed);
      str_cat(dstemp,"  Enemies killed");
      print(dstemp);
      waitt(40);
      my_text.visible = off;

      calculate_time();
      //creating new 5 ENEMIES after 10 ENEMIES are died
      if(ene_killed == 10)
      {
```

```
            create_ene();
            create_ene();
            create_ene();
            create_ene();
            create_ene();
            eneescape_time = count_time;
          }

          ene_killtry += 1;
          baby_armor += 5 + int(random(10));
          baby_power += 3;
          baby_health -= 5 + int(random(10));

          if((count_time - eneescape_time) > 300 )
          {
            if(ene_killed < 15)
            {
              eindex = 0;
              create_enemy();
              fort_captured = 0;
              ene_need_kill = 25;
              beep();
              print("Fort was recaptured by the enemies");
              waitt(40);
              my_text.visible = off;
            }
            else
            {
              if(ene_killed == ene_need_kill)
              {
                fort_captured = 1;
                fortcaptured_time = count_time;
                beep();
                print("Congratualations! Fort was captured");
                waitt(40);
                my_text.visible = off;
              }
            }
          }
        }

      }
      waitt(8);
    }
}

function create_ene()
{
  if(eindex < 30)
  {
    temp.x = e_start_pos[eindex];
    temp.y = e_start_pos[eindex + 1];
    temp.z = e_start_pos[eindex + 2];
    eindex += 3;
  }
  else
  {
    temp.x = 730 + RANDOM(500);
    temp.y = 4350 + RANDOM(700);
    temp.z =  200;
  }
  ene_entity = ent_create(enemy_mdl,temp,aenemy);
  ene_entity.skill11 = 1;
  ene_entity.skill33 = HANDLE(ene_entity);
```

```
  print("Enemy created");
  waitt(40);
  my_text.visible = off;
}

function create_enemy()
{
  create_ene();
  create_ene();
  create_ene();
  create_ene();
  create_ene();
  create_ene();
  create_ene();
  create_ene();
  create_ene();
  create_ene();
}
```

Healthpac.wdl

```
var h_start_pos[90] = 6275, -7150, 120, -5700, -8150, 70, -7300, 4700, 70, 6000, 4750, 50;
var hindex = 0;

entity* hea_entity;
string healthpac_mdl = <healthpac.mdl>;

action ahealthpac
{
  my.enable_scan = on;       // sensible of explosions
  my.enable_detect = on; // sensible of the player
}

//creating HEALTHPACs at predefined locations
function create_health()
{
  temp.x = h_start_pos[hindex];
  temp.y = h_start_pos[hindex + 1];
  temp.z = h_start_pos[hindex + 2];
  hindex += 3;

  hea_entity = ent_create (healthpac_mdl, temp, ahealthpac);
  hea_entity.skill7 = 1;
  hea_entity.skill21 = handle(hea_entity);
}

//creating 4 HEALTHPACs
function create_healthpac()
{
  create_health();
  create_health();
  create_health();
  create_health();
}
```

Armorpac.wdl

```
var a_start_pos[90] = 7000,-6700,250, 6600,-7700,550, 1000,7300,50, -4000,7000,50, 1000,4350,100, -5200,-2200,200;
var aindex = 0;
```

```
entity* arm_entity;
string armorpac_mdl = <armorpac.mdl>;

action aarmorpac
{
  my.enable_scan = on;       // sensible of explosions
  my.enable_detect = on; // sensible of the player
}

//creating ARMORPACs at predefined locations
function create_armor()
{
  temp.x = a_start_pos[aindex];// + random(25);
  temp.y = a_start_pos[aindex + 1];// + random(15);
  temp.z = a_start_pos[aindex + 2];// + random(5);
  aindex += 3;

  arm_entity = ent_create (armorpac_mdl, temp, aarmorpac);
  arm_entity.skill9 = 1;
  arm_entity.skill27 = handle(arm_entity);
}

//creating 6 ARMORPACS
function create_armorpac()
{
  create_armor();
  create_armor();
  create_armor();
  create_armor();
  create_armor();
  create_armor();
}
```

Priest.wdl

```
var p_start_pos[90] = 6300, 1100, 60, 3000, -2000, 60, 1100, -6800, 60, -5000, 0, 60, -5700, 4000, 60;
var pindex = 0;

entity* pre_entity;
string priest_mdl = <priest.mdl>;

action apriest
{
  my.enable_scan = on;       // sensible of explosions
  my.enable_detect = on; // sensible of the player
}

//placing PRIESTs at random places at a location
function create_pre()
{
  temp.x = -7225 + random(300);
  temp.y = 4300 + random(400);
  temp.z = 70;

  pre_entity = ENT_CREATE (priest_mdl, temp, apriest);
  pre_entity.skill13 = 1;
  pre_entity.skill39= HANDLE(pre_entity);
  print("Priest created");
  waitt(40);
  my_text.visible = off;
}

//creating 2 PRIESTs
```

```
function create_priest()
{
  create_pre();
  create_pre();
}
```

Datastruct.wdl

```
//variables for inserting function
var model_num;
var action_num;
var time_var;
var combo;

var gold;
var power;
var armor;
var health;

//variables to deceide entries
var max_entries = 10;
var items = 4;
var start = 0;
var loc[40];
var orderloc[40];
var stacktop = 0;
var toppointer = 0;

//variables to count for models encountered and actions perfrmed
var mon_val = 0;
var sam_val = 0;
var vil_val = 0;
var pre_val = 0;
var ene_val = 0;
var hea_val = 0;
var arm_val = 0;

var kill_val = 0;
var hire_val = 0;
var pick_val = 0;
var stay_val = 0;

var moki_val = 0;
var saki_val = 0;
var sahi_val = 0;
var sast_val = 0;
var viki_val = 0;
var vihi_val = 0;
var vist_val = 0;
var prki_val = 0;
var prhi_val = 0;
var prst_val = 0;
var enki_val = 0;
var hepi_val = 0;
var hest_val = 0;
var arpi_val = 0;
var arst_val = 0;

var eventcounter = 6;

function insert(model_num, action_num, combo, time_var)
{
  eventcounter += 1;
```

```
  if(start >= 40)//max_entries * 4) // 3->is total #  of parameters => items
  {
    start = 0;
  }
  STACKTOP = start;

  LOC[start] = model_num;
  LOC[start + 1] = action_num;
  LOC[start + 2] = combo;
  LOC[start + 3] = time_var;
  start += items;
}
string mystr;
string strnum;

var locnum = 0;
var counter = 0;

function insertshow()
{
  locnum = 0;
  TOPPOINTER = STACKTOP;
  while(locnum < 40) //15-> is total # of entries => max_entries*items
  {
    if(counter != 4) // 3->is total # of parameters => items
    {
      str_for_num(strnum, LOC[TOPPOINTER]);
      str_cat(mystr,strnum);
      print(mystr);
      waitt(20);
      my_text.visible = off;
      counter += 1;
      TOPPOINTER += 1;
      locnum += 1;
    }
    ELSE
    {
      counter = 0;
      if(TOPPOINTER != 4) // 3->is total # of parameters => items
      {
        TOPPOINTER -= 8; // 6->is items * 2
      }
      else
      {
        TOPPOINTER = 36; //12-> is (max_entries-1) * (items)
      }
    }
  }
  counter = 0;
  str_cpy(mystr," ");
}

function order_arrange()
{
  locnum = 0;
  counter = 0;
  str_cpy(mystr," ");
  TOPPOINTER = STACKTOP;
  while(locnum < 40) //15-> is total # of entries => max_entries*items
  {
    if(counter != 4) // 3->is total # of parameters => items
    {
      orderloc[locnum] = loc[TOPPOINTER];
      counter += 1;
      TOPPOINTER += 1;
```

```
      locnum += 1;
    }
    ELSE
    {
      counter = 0;
      if(TOPPOINTER != 4) // 3->is total # of parameters => items
      {
        TOPPOINTER -= 8; // 6->is items * 2
      }
      else
      {
        TOPPOINTER = 36; //12-> is (max_entries-1) * (items)
      }
    }
  }
}

function reset_MODEL()
{
  mon_val = 0;
  sam_val = 0;
  vil_val = 0;
  pre_val = 0;
  ene_val = 0;
  hea_val = 0;
  arm_val = 0;
}

function reset_ACTION()
{
  kill_val = 0;
  hire_val = 0;
  pick_val = 0;
  stay_val = 0;
}

function reset_combo()
{
  moki_val = 0;
  saki_val = 0;
  sahi_val = 0;
  sast_val = 0;
  viki_val = 0;
  vihi_val = 0;
  vist_val = 0;
  prki_val = 0;
  prhi_val = 0;
  prst_val = 0;
  enki_val = 0;
  hepi_val = 0;
  hest_val = 0;
  arpi_val = 0;
  arst_val = 0;
}


string model_string = "Not yet Known";
var mmaxval = 0;
var mival = 0;
var mtempval = 0;

string action_string = "Not yet Known";
var amaxval = 0;
var aival = 0;
var atempval = 0;
```

```
function find_maxModel()
{
  reset_MODEL();
  order_arrange();

  mival = 0;
  mmaxval = 0;

  while(mival < 10) //5-> is max_entries
  {
    mtempval = mival*4; // 3->is total # of parameters => items
    if(ORDERLOC[mtempval] == 1)
    {
      mon_val += 1;
    }
    if(ORDERLOC[mtempval] == 2)
    {
      sam_val += 1;
    }
    if(ORDERLOC[mtempval] == 3)
    {
      vil_val += 1;
    }
    if(ORDERLOC[mtempval] == 4)
    {
      pre_val += 1;
    }
    if(ORDERLOC[mtempval] == 5)
    {
      ene_val += 1;
    }
    if(ORDERLOC[mtempval] == 6)
    {
      hea_val += 1;
    }
    if(ORDERLOC[mtempval] == 7)
    {
      arm_val += 1;
    }
    mival += 1;

  }

  if(mon_val > mmaxval)
  {
    mmaxval = mon_val;
    str_cpy(model_string,"MONSTOR");
  }
  if(mmaxval < sam_val)
  {
    mmaxval = sam_val;
    str_cpy(model_string,"SAMURAI");
  }

  if(mmaxval < vil_val)
  {
    mmaxval = vil_val;
    str_cpy(model_string,"VILLAGER");
  }
  if(mmaxval < pre_val)
  {
    mmaxval = pre_val;
    str_cpy(model_string,"PRIEST");
  }
```

```
  if(mmaxval < ene_val)
  {
    mmaxval = ene_val;
    str_cpy(model_string,"ENEMY");
  }
  if(mmaxval < hea_val)
  {
    mmaxval = hea_val;
    str_cpy(model_string,"HEALTHPAC");
  }
  if(mmaxval < arm_val)
  {
    mmaxval = arm_val;
    str_cpy(model_string,"ARMORPAC");
  }
}

function find_maxAction()
{
  reset_ACTION();
  order_arrange();

  aival = 0;
  amaxval = 0;

  while(aival < 10)  //5-> is max_entries
  {
    atempval = aival * 4 + 1;// 3->is total # of parameters => items
    if(ORDERLOC[atempval] == 1)
    {
      kill_val += 1;
    }
    if(ORDERLOC[atempval] == 2)
    {
      hire_val += 1;
    }
    if(ORDERLOC[atempval] == 3)
    {
      pick_val += 1;
    }
    if(ORDERLOC[atempval] == 4)
    {
      stay_val += 1;
    }

    aival += 1;
  }

  if(kill_val > amaxval)
  {
    amaxval = kill_val;
    str_cpy(action_string,"KILL");
  }
  if(amaxval < hire_val)
  {
    amaxval = hire_val;
    str_cpy(action_string,"HIRE");
  }

  if(amaxval < pick_val)
  {
    amaxval = pick_val;
    str_cpy(action_string,"PICK");
  }
  if(amaxval < stay_val)
```

```
  {
    amaxval = stay_val;
    str_cpy(action_string,"STAY");
  }

  aival = 0;
}

var cival = 0;
var cmaxval = 0;
var ctempval = 0;
string combo_string;

function find_maxCombo()
{
  reset_COMBO();
  order_arrange();

  cival = 0;
  cmaxval = 0;

  while(cival < 10)  //5-> is max_entries
  {
    ctempval = cival*4 + 2;   // 3->is total # of parameters => items
    if(ORDERLOC[ctempval] == 1)
    {
      moki_val += 1;
    }
    if(ORDERLOC[ctempval] == 2)
    {
      saki_val += 1;
    }
    if(ORDERLOC[ctempval] == 3)
    {
      sahi_val += 1;
    }
    if(ORDERLOC[ctempval] == 4)
    {
      sast_val += 1;
    }
    if(ORDERLOC[ctempval] == 5)
    {
      viki_val += 1;
    }
    if(ORDERLOC[ctempval] == 6)
    {
      vihi_val += 1;
    }
    if(ORDERLOC[ctempval] == 7)
    {
      vist_val += 1;
    }
    if(ORDERLOC[ctempval] == 8)
    {
      prki_val += 1;
    }
    if(ORDERLOC[ctempval] == 9)
    {
      prhi_val += 1;
    }
    if(ORDERLOC[ctempval] == 10)
    {
      prst_val += 1;
    }
    if(ORDERLOC[ctempval] == 11)
```

```
  {
    enki_val += 1;
  }
  if(ORDERLOC[ctempval] == 12)
  {
    hepi_val += 1;
  }
  if(ORDERLOC[ctempval] == 13)
  {
    hest_val += 1;
  }
  if(ORDERLOC[ctempval] == 14)
  {
    arpi_val += 1;
  }
  if(ORDERLOC[ctempval] == 15)
  {
    arst_val += 1;
  }
  cival += 1;
}

if(moki_val > cmaxval)
{
  cmaxval = moki_val;
  str_cpy(COMBO_string,"KILLING MONSTOR");
}
if(cmaxval < saki_val)
{
  cmaxval = saki_val;
  str_cpy(combo_string,"KILLING SAMURAI");
}
if(cmaxval < sahi_val)
{
  cmaxval = sahi_val;
  str_cpy(combo_string,"HIRING SAMURAI");
}if(cmaxval < sast_val)
{
  cmaxval = sast_val;
  str_cpy(combo_string,"STAYING AT SAMURAI");
}
if(cmaxval < viki_val)
{
  cmaxval = viki_val;
  str_cpy(combo_string,"KILLING VILLAGER");
}
if(cmaxval < vihi_val)
{
  cmaxval = vihi_val;
  str_cpy(combo_string,"HIRING VILLAGER");
}
if(cmaxval < vist_val)
{
  cmaxval = vist_val;
  str_cpy(combo_string,"STAYING AT VILLAGER");
}
if(cmaxval < prki_val)
{
  cmaxval = prki_val;
  str_cpy(combo_string,"KILLING PRIEST");
}
if(cmaxval < prhi_val)
{
  cmaxval = prhi_val;
  str_cpy(combo_string,"ASKING HELP FROM PRIEST");
```

```
  }
  if(cmaxval < prst_val)
  {
    cmaxval = prst_val;
    str_cpy(combo_string,"STAYING AT PRIESR");
  }
  if(cmaxval < enki_val)
  {
    cmaxval = enki_val;
    str_cpy(combo_string,"KILLING ENEMIES");
  }
  if(cmaxval < hepi_val)
  {
    cmaxval = hepi_val;
    str_cpy(combo_string,"PICKUP THE HEALTHPAC");
  }if(cmaxval < hest_val)
  {
    cmaxval = hest_val;
    str_cpy(combo_string,"STAYING AT HEALTHPAC");
  }
  if(cmaxval < arpi_val)
  {
    cmaxval = arpi_val;
    str_cpy(combo_string,"PICKUP THE ARMORPAC");
  }if(cmaxval < arst_val)
  {
    cmaxval = arst_val;
    str_cpy(combo_string,"STAYING AT ARMORPAC");
  }
}


function analysis()
{
  find_maxModel();
  waitt(20);
  find_maxAction();
  waitt(20);
  find_maxCombo();
}

on_a = find_maxAction();
on_m = find_maxModel();
on_o = order_arrange();
on_z = insertshow();
on_q = analysis();
```

Rulebase.wdl

```
var goal1 = 0;
var goal2 = 0;
var goal3 = 0;
var goal4 = 0;
var goal5 = 0;

function rulebase_recent()
{
analysis();

if(str_cmpi(action_string, "PICK"))
 {
  if(moki_val > 1)
  {
    goal4 = 0;
```

```
}

print("You are inside pick");
waitt(40);

if(!((str_cmpi(model_string, "HEALTHPAC")) || (str_cmpi(model_string, "ARMORPAC"))))// checking for models not match
// with action
{
  if(str_cmpi(model_string, "MONSTER"))
  {
    goal3  += (moki_val - sahi_val);
    goal5 += (sahi_val - saki_val);
  }
  if(str_cmpi(model_string, "ENEMY"))
  {
    goal2 += (enki_val * 2);  // may be we can change the constant from 2 to 3 or more
  }
  if(str_cmpi(model_string, "SAMURAI"))
  {
    if(sahi_val > 2)
    {
      goal5 += sahi_val;
    }
    else
    {
      if(sast_val > saki_val)
      {
        goal4 += sast_val;
      }
      else
      {
        goal3 += saki_val;
      }
    }
  }
  if(str_cmpi(model_string, "VILLAGER"))
  {
    if(vihi_val > 2)
    {
      goal1 += vihi_val;
    }
    else
    {
      if(vist_val > viki_val)
      {
        goal4 += vist_val;
      }
      else
      {
        goal1 -= viki_val;
        goal3 += viki_val;
      }
    }
  }
  if(str_cmpi(model_string, "PRIEST"))
  {
    if(prhi_val > 1)
    {
      goal5 += (prhi_val * 2);
    }
    else
    {
      if(prst_val > prki_val)
      {
        goal4 += prst_val;
```

```
        }
        else
        {
          goal3 += prki_val;
          goal5 -= (prki_val * 2);
        }
      }
    }
  }
  else  // models that match with the action "PICK"
  {
    if(str_cmpi(model_string, "HEALTHPAC"))  // deal more with healthpacs
    {
      if(hea_val >2)//healthpacs picked in the last 10 actions
      {
        if(ene_val || mon_val)//dealing with enemies or monsters in the last 10 actions
        {
          if(ene_val > mon_val)
          {
            goal2 += (ene_val * 2);
          }
          else
          {
            goal3 += (moki_val + saki_val + viki_val - sahi_val - vihi_val);
            goal5 += sahi_val;
          }
        }
        else //dealing with objects except monster & enemies
        {
          goal4 += 2;
          if((baby_health - (hea_val * 15)) < 25 )// checking if the user is picking
          // because of less health
          {
            goal1 += (vihi_val * 2);
            goal2 += (enki_val * 2);
            goal3 += (moki_val - vihi_val);
            goal5 += (sahi_val + (prhi_val * 2) + (knighthood * 2));
          }
        }
      }
    }

    if(str_cmpi(model_string, "ARMORPAC"))// deal more with armorpacs
    {
      if(arm_val > 2)//armorpacs picked in the last 10 actions
      {
        if((baby_armor - (arm_val * 15)) < 25 )// checking if the user is picking
        // because of less armor
        {
          goal3 += arpi_val;
        }
        if(sam_val || mon_val)//dealing with samurais or monsters in the last 10 actions
        {
          if(sam_val > mon_val)
          {
            goal5 += (sam_val - mon_val);
          }
          else
          {
            goal3 += (moki_val + saki_val + viki_val - sahi_val - vihi_val);
            goal5 += (sam_val - mon_val);
          }
        }
        else //dealing with objects except monster & samurais
        {
```

```
            goal4 += 2;
          }
        }
      }
    }
  }

  //  analysis for "PICK" ends here

  //  analysis for "STAY" starts here

 if(str_cmpi(action_string, "STAY"))
  {
   print("You are inside stay");
   waitt(40);

   if((str_cmpi(model_string,"MONSTER")) || (str_cmpi(model_string,"ENEMY"))) // contract combination of action & models
   {
     if(moki_val > enki_val)
     {
       goal2 += enki_val;
       goal3 += (moki_val + saki_val + viki_val - sahi_val - vihi_val);
     }
     else
     {
       goal2 += enki_val * 2;
     }
   }
   else  //checking for possible combination of action "STAY" and models
   {
     if(str_cmpi(model_string, "SAMURAI"))  // dealt with "SAMURAI" mostly
     {
       if(sast_val >1)  // stayed with "SAMURAI" for sufficient # of times
       {
         goal4 += sast_val;
       }
       else
       {
         if(sahi_val > saki_val)  //otherwise checking the interactions with "SAMURAI"
         {
           goal5 += sahi_val;
         }
         else
         {
           goal3 += saki_val;
         }
       }
     }
     if(str_cmpi(model_string, "VILLAGER")) // dealt with "VILLAGER" mostly
     {
       if(vist_val > 1)// stayed with "VILLAGER" for sufficient # of times
       {
         goal4 += vist_val;
       }
       else
       {
         if(vihi_val > viki_val) //otherwise checking the interactions with "VILLAGER"
         {
           goal1 += vihi_val;
         }
         else
         {
           goal1 -= viki_val;
           goal3 += (viki_val * 2);
         }
```

```
      }
    }
    if(str_cmpi(model_string, "PRIEST"))   // dealt with "PRIEST" mostly
    {
      if(prst_val > 1)  // stayed with "PRIEST" for sufficient # of times
      {
        goal4 += prst_val;
      }
      else
      {
        if(prhi_val > prki_val)//otherwise checking the interactions with "PRIEST"
        {
          goal5 += prhi_val + sahi_val;
        }
        else
        {
          goal3 += (prki_val * 2);
          goal5 -= (prki_val * 3);
        }
      }
    }
    if(str_cmpi(model_string, "ARMORPAC"))// dealt with "ARMORPAC" mostly
    {
      if(arpi_val > arst_val)   // checking the interactions with "ARMORPAC"
      {
        goal3 += moki_val;
      }
      else
      {
        goal4 += (arst_val - moki_val);
      }
    }
    if(str_cmpi(model_string, "HEALTHPAC"))  // dealt with "HEALTHPAC" mostly
    {
      if(hepi_val > hest_val) // checking the interactions with "HEALTHPAC"
      {
        goal3 += hepi_val;
        goal5 += moki_val;
      }
      else
      {
        goal4 += hest_val;
      }
    }
  }
 }

 //   analysis for "STAY" ends here

 //   analysis for "HELP" starts here

if(str_cmpi(action_string, "HIRE"))
 {
  print("You are inside Hire");
  waitt(40);

  if((str_cmpi(model_string,"MONSTER")) || (str_cmpi(model_string,"ENEMY"))) // contrast combination of action & models
  {
    if(moki_val > enki_val)
    {
      goal5 += (moki_val + sahi_val - vihi_val);
    }
    else
    {
      goal2 += enki_val;
```

```
  }
}
else  // checking for possible combinations of action "HELP" and models
{
  if(str_cmpi(model_string, "SAMURAI"))
  {
    if(sahi_val > 2)
    {
      goal5 += sahi_val;
    }
    else
    {
      if(sahi_val > saki_val)
      {
        goal5 += sahi_val;
      }
      else
      {
        goal3 += saki_val;
      }
    }
  }
  if(str_cmpi(model_string, "VILLAGER"))
  {
    if(vihi_val > 1)
    {
      goal1 += vihi_val;
    }
    else
    {
      if(vihi_val > viki_val)
      {
        goal1 += vihi_val;
      }
      else
      {
        goal3 += viki_val;
      }
    }
  }
  if(str_cmpi(model_string, "PRIEST"))
  {
    if(prhi_val)
    {
      goal5 += (sahi_val + (prhi_val * 2));
    }
    else
    {
      if(prst_val > prki_val)
      {
        goal4 += prst_val;
      }
      else
      {
        goal3 += prki_val;
        goal5 -= prki_val;
      }
    }
  }
  if((str_cmpi(model_string, "ARMORPAC")) || (str_cmpi(model_string, "HEALTHPAC"))) // these models can be picked only
  {
    goal1 += vihi_val;
    goal2 += enki_val;
    goal3 += (moki_val + saki_val + viki_val - sahi_val - vihi_val);
    goal4 += (arpi_val + hepi_val - saki_val - viki_val - enki_val);
```

```
      goal5 += (moki_val + sahi_val);
    }
  }
 }

 //  analysis for "HELP" ends here

 //  analysis for "KILL" starts here

if(str_cmpi(action_string, "KILL"))
 {
  print("You are inside kill");
  waitt(40);

  goal1 = 0;

  if((str_cmpi(model_string,"HEALTHPAC")) || (str_cmpi(model_string,"ARMORPAC")))
  {
    if(viki_val || saki_val)
    {
      goal3 += (viki_val + saki_val);
    }
  }
  else
  {
    if(str_cmpi(model_string, "MONSTER"))
    {
      if(moki_val > 2)
      {
        goal3 += moki_val;// + saki_val + viki_val - sahi_val - vihi_val);
        goal5 += sahi_val;
      }
      else
      {
        goal5 += (moki_val + sahi_val);
      }
    }
    if(str_cmpi(model_string, "ENEMY"))
    {
      if(enki_val > 2)
      {
        goal2 += (enki_val * 2);
      }
      else
      {
        goal3 += enki_val;
      }
    }
    if(str_cmpi(model_string, "SAMURAI"))
    {
      if(saki_val > 1)
      {
        goal3 += saki_val;
        goal5 -= (saki_val * 3);
      }
      else
      {
        if(sahi_val > sast_val)
        {
          goal5 += sahi_val;
        }
        else
        {
          goal4 += sast_val;
        }
```

```
      }
    }
    if(str_cmpi(model_string, "VILLAGER"))
    {
      if(viki_val > 1)
      {
        goal1 -= viki_val;
        goal3 += viki_val;
      }
      else
      {
        if(vihi_val > vist_val)
        {
          goal1 += vihi_val;
        }
        else
        {
          goal4 += vist_val;
        }
      }
    }
    if(str_cmpi(model_string, "PRIEST"))
    {
      if(prki_val)
      {
        goal3 += (prki_val * 2);
        goal5 -= (prki_val * 2);
      }
      else
      {
        if(prhi_val > prst_val)
        {
          goal5 += (sahi_val + moki_val);
        }
        else
        {
          goal4 += (prst_val * 2);
        }
      }
    }
  }
 }

 //  analysis for "KILL" ends here

if((viki_val >= vihi_val) || (viki_val > 2))
 {
  goal1 = 0;
 }
if((moki_val + saki_val + viki_val) < 2)
 {
  goal3 = 0;
 }
if(saki_val || viki_val)
 {
  goal4 = 0;
 }
if(smonster_dead ||(vihi_val > 3) || prki_val)
 {
  goal5 = 0;
 }
if((moki_val - sahi_val) > 4)
 {
  goal3 += moki_val;
  goal5 = 0;
```

```
 }

if((sam_hfailed || vil_hfailed) && moki_val)
 {
  if(sam_hftime > vil_hftime)
  {
    if(knight_knowledge)
    {
      goal5 += 5;
    }
    else
    {
      goal3 += 5;
    }
  }
  else
  {
    goal1 += 5;
  }
 }

}//  function ends here

var mon_count = 1;
var ene_count = 1;
var sam_kcount = 1;
var sam_hcount = 1;
var vil_kcount = 1;
var vil_hcount = 1;
var pre_kcount = 1;
var pre_hcount = 1;
var arm_count = 1;
var hea_count = 1;


var rbtemp1 = 0;
var rbtemp2 = 0;
var rbtemp3 = 0;
var rbtemp4 = 0;
var rbtemp5 = 0;
var rbtemp6 = 0;
var rbtemp7 = 0;
var rbtemp8 = 0;

function rulebase_history()
{
analysis();
if(mon_killed > (3 * mon_count))  // checking for monster killings
 {
  if(sam_hired > mon_killed)
  {
    goal5 += sahi_val;
  }
  if(vil_hired > mon_killed)
  {
    goal1 += vihi_val;
  }
  if(!(sahi_val || vihi_val || enki_val))
  {
    goal3 += 3;
  }
  mon_count += 1;
 }

if(ene_killed > (3 * ene_count)) // checking for enemy killings
```

```
 {
  goal2 += 5 + enki_val;
  ene_count += 1;
 }

if(sam_killed > (3 * sam_kcount))// checking for samurai killings
 {
  goal3 += saki_val;
  sam_kcount += 1;
 }

if(sam_hired > (3 * sam_hcount))// checking for samurai hirings
 {
  goal5 += sahi_val;
  sam_hcount += 1;
 }
if(vil_killed > (2 * vil_kcount))  // checking for villager killings
 {
  goal1 -= (viki_val * 2);
  goal3 += viki_val;
  vil_kcount += 1;
 }

if(vil_hired > (3 * vil_hcount))  // checking for villager hirings
 {
  goal1 += vihi_val;
  vil_hcount += 1;
 }
if(pre_killed > pre_kcount)// checking for priest killings
 {
  goal3 += 2;
  goal5 -= 2;
  pre_kcount += 1;
 }

if(pre_helptry > pre_hcount)// checking for priest hirings
 {
  goal5 += 3;
  pre_hcount += 1;
 }
if(arm_picked > (3 * arm_count)) // checking for armorpacs
 {
  if((moki_val)&&(sahi_val || vihi_val))
  {
    if(sahi_val > vihi_val)
    {
      goal5 += sahi_val;
    }
    else
    {
      goal1 += vihi_val;
    }
  }
  else
  {
    goal3 += max(arpi_val, 3);
  }
  arm_count += 1;
 }

if(hea_picked > (3 * hea_count)) // checking for healthpacs
 {
  if((moki_val)&&(sahi_val || vihi_val))
  {
    if(sahi_val > vihi_val)
```

```
    {
      goal5 += sahi_val;
    }
    else
    {
      goal1 += vihi_val;
    }
  }
  else
  {
    goal4 += max(hepi_val, 3);
  }
  hea_count += 1;
 }

 //  world states or world state knowledge

//***** "KNIGHT KNOWLEDGE" starts here*****

if(knight_knowledge && !(knighthood))
 {
  if(sam_hired > 4)
  {
    goal5 += sahi_val;
  }
  else
  {
    goal5 -= 1;
  }

  if(saki_val)
  {
    while(rbtemp1 < 10)
    {
      if(orderloc[(rbtemp1*4 + 2)] == 2)// checking for saki_val
      {
        if(orderloc[(rbtemp1*4 + 3)] - knightknow_time) // saki after knight knowledge
        {
          goal3 += saki_val;
          goal5 -= saki_val;
          rbtemp1 = 10;
        }
        else
        {
          rbtemp1 = 10;
        }
      }
      rbtemp1 += 1;
      waitt(1);
    }
    rbtemp1 = 0;
  }

  if(moki_val)
  {
    while(rbtemp1 < 10)
    {
      if(orderloc[(rbtemp1*4 + 2)] == 1)// checking for moki_val
      {
        if(orderloc[(rbtemp1*4 + 3)] - knightknow_time) // moki after knight knowledge
        {
          if(sahi_val)  // hired samurais
          {
            goal5 += sahi_val;
          }
```

```
            rbtemp1 = 10;
          }
          else
          {
            rbtemp1 = 10;
          }
        }
        rbtemp1 += 1;
        waitt(1);
      }
      rbtemp1 = 0;
    }
  }

//***** "KNIGHT KNOWLEDGE" ends here*****

//***** "KNIGHTHOOD" starts here*****

if(knighthood)
 {
  if(sahi_val)
  {
    while(rbtemp2 < 10)
    {
      if(orderloc[(rbtemp2*4 + 2)] == 3)// checking for sahi_val
      {
        if(orderloc[(rbtemp2*4 + 3)] - knighthood_time) // sahi after knighthood
        {
          if(enemy_knowledge)
          {
            goal2 += (sahi_val + enki_val);
          }
          else
          {
            goal3 += sahi_val;
          }
          rbtemp2 = 10;
        }
        else
        {
          rbtemp2 = 10;
        }
      }
      rbtemp2 += 1;
      waitt(1);
    }
    rbtemp2 = 0;
  }

  if(saki_val)
  {
    while(rbtemp2 < 10)
    {
      if(orderloc[(rbtemp2*4 + 2)] == 2)// checking for saki_val
      {
        if(orderloc[(rbtemp2*4 + 3)] - knighthood_time) // saki after knighthood
        {
          goal3 += saki_val;
          goal5 = 0;//saki_val;
          rbtemp2 = 10;
        }
        else
        {
          rbtemp2 = 10;
        }
```

```
      }
      rbtemp2 += 1;
      waitt(1);
    }
    rbtemp2 = 0;
  }

  if(moki_val)
  {
    while(rbtemp2 < 10)
    {
      if(orderloc[(rbtemp2*4 + 2)] == 1)// checking for moki_val
      {
        if(orderloc[(rbtemp2*4 +3)] - knighthood_time)  // moki after knighthood
        {
          if(dragon_knowledge && !(smonster_dead))
          {
            goal5 += max(sahi_val, moki_val);
          }

          rbtemp2 = 10;
        }
        else
        {
          rbtemp2 = 10;
        }
      }
      rbtemp2 += 1;
      waitt(1);
    }
    rbtemp2 =0;
  }

  if(enki_val)
  {
    goal2 += enki_val;
  }
 }

//***** "KNIGHTHOOD" ends here*****


//***** "TOWN KNOWLEDGE" starts here*****

if(town_knowledge && !(town_built))
 {
  if(vihi_val) //after knowledge
  {
    while(rbtemp3 < 10 )
    {
      if(orderloc[(rbtemp3*4 + 2)] == 6)// checking for vihi_val
      {
        if(orderloc[(rbtemp3*4 + 3)] - townknow_time)   // vihi after town knowledge
        {
          goal1 += (vihi_val * 2);
          rbtemp3 = 10;
        }
        else
        {
          rbtemp3 = 10;
        }
      }
      rbtemp3 += 1;
      waitt(1);
```

```
    }
    rbtemp3 = 0;
  }
  else  // not hired villagers
  {
    goal1 = 0;
  }
  if(viki_val) //after knowledge
  {
    while(rbtemp3 < 10)
    {
      if(orderloc[(rbtemp3*4 + 2)] == 5)// checking for viki_val
      {
        if(orderloc[(rbtemp3*4 + 3)] - townknow_time)   // viki after town knowledge
        {
          goal1 -= (viki_val * 2);
          goal3 += viki_val;
          rbtemp3 = 10;
        }
        else
        {
          rbtemp3 = 10;
        }
      }
      rbtemp3 += 1;
      waitt(1);
    }
    rbtemp3 = 0;
  }

  if(moki_val)
  {
    if(sahi_val > vihi_val)   // check if he hire more samurais than villagers
    {
      goal5 += sahi_val;
    }
    else
    {
      goal1 += sahi_val;
    }
  }
 }

//***** "TOWN KNOWLEDGE" ends here*****

//***** "TOWN BUILT" starts here*****

if(town_built)
 {
  if(vihi_val) // check for hiring after town built
  {
    while(rbtemp4 < 10)
    {
      if(orderloc[(rbtemp4*4 + 2)] == 6)// checking for vihi_val
      {
        if(orderloc[(rbtemp4*4 + 3)] - townbuilt_time)  // vihi after town built
        {
          goal4 += vihi_val;
          rbtemp4 = 10;
        }
        else
        {
          rbtemp4 = 10;
        }
      }
```

```
      rbtemp4 += 1;
      waitt(1);
    }
    rbtemp4 = 0;
  }
 }

//***** "TOWN BUILT" ends here*****

//***** "ENEMY KNOWLEDGE" starts here*****

if(enemy_knowledge && !(fort_captured))
 {
  if(!(knighthood))
  {
    if(sahi_val)
    {
      while(rbtemp5 < 10)
      {
        if(orderloc[(rbtemp5*4 + 2)] == 3)// checking for sahi_val
        {
          if(orderloc[(rbtemp5*4 + 3)] - enemyknow_time)  // sahi after knighthood
          {
            goal2 += (sahi_val + enki_val);
            goal5 += sahi_val;
            rbtemp5 = 10;
          }
          else
          {
            rbtemp5 = 10;
          }
        }
        rbtemp5 += 1;
        waitt(1);
      }
      rbtemp5 =0;
    }
  }
  else  // user is knight here
  {
    if(sahi_val)
    {
      while(rbtemp5 < 10)
      {
        if(orderloc[(rbtemp5*4 + 2)] == 3)// checking for sahi_val
        {
          if(orderloc[(rbtemp5*4 + 3)] - enemyknow_time)  // sahi after knighthood
          {
            goal2 += sahi_val;
            rbtemp5 = 10;
          }
          else
          {
            rbtemp5 = 10;
          }
        }
        rbtemp5 += 1;
        waitt(1);
      }
      rbtemp5 =0;
    }
  }

  if(moki_val)
  {
```

```
    if((moki_val * 20) > baby_gold) // gold is spent for some thing
    {
      if(sahi_val > vihi_val)
      {
        goal2 += sahi_val;
        goal5 += sahi_val;
      }
    }
  }

  if(enki_val)
  {
    goal2 += (enki_val * 2);
  }
 }

//***** "ENEMY KNOWLEDGE" ends here*****

//***** "ENEMY CAPTURED" starts here*****

if(fort_captured)
 {
  if(vihi_val)
  {
    while(rbtemp6 < 10)
    {
      if(orderloc[(rbtemp6*4 + 2)] == 6)// checking for vihi_val
      {
        if(orderloc[(rbtemp6*4 + 3)] - fortcaptured_time)  // vihi after fort was captured
        {
          if(town_built)
          {
            goal4 += vihi_val;
          }
          else
          {
            goal1 += vihi_val;
          }
          rbtemp6 = 10;
        }
        else
        {
          rbtemp6 = 10;
        }
      }
      rbtemp6 += 1;
      waitt(1);
    }
    rbtemp6 = 0;
  }

  if(moki_val)
  {
    while(rbtemp6 < 10)
    {
      if(orderloc[(rbtemp6*4 + 2)] == 1)// checking for moki_val
      {
        if(orderloc[(rbtemp6*4 + 3)] - fortcaptured_time)  // moki after fort was captured
        {
          if(!(smonster_dead))
          {
            goal5 += moki_val;
          }
          else
          {
```

```
            goal3 += moki_val;
          }
          rbtemp6 = 10;
        }
        else
        {
          rbtemp6 = 10;
        }
      }
      rbtemp6 += 1;
      waitt(1);
    }
    rbtemp6 = 0;
  }

  if(saki_val || viki_val)
  {
    goal3 += (saki_val + viki_val);
  }
 }

//***** "ENEMY CAPTURED" ends here*****

//***** "S_MONSTER KNOWLEDGE" starts here*****

if(dragon_knowledge && !(smonster_dead))
 {
  if(moki_val)
  {
    while(rbtemp7 < 10)
    {
      if(orderloc[(rbtemp7*4 + 2)] == 1)// checking for moki_val
      {
        if(orderloc[(rbtemp7*4 + 3)] - smonsterknow_time) // moki after knighthood
        {
          if(!(knighthood))
          {
            if(sahi_val)
            {
              goal2 += (sahi_val + enki_val);
              goal5 += (sahi_val + moki_val);
            }
          }
          else  // user is knight here
          {
            goal3 += moki_val;
            goal5 += (moki_val * 2);
          }
          rbtemp7 = 10;
        }
        else
        {
          rbtemp7 = 10;
        }
      }
      rbtemp7 += 1;
      waitt(1);
    }
    rbtemp7 = 0;
  }

  if(sahi_val)
  {
    if(knighthood)
    {
```

```
      goal2 += (sahi_val + enki_val);
      goal5 += (sahi_val + moki_val);
    }
  }
 }
}

var gmaxval = 0;
string goal_string[90];
string print_goal[90];
function find_maxGoal()
{
str_cpy(goal_string,"NO Information");
gmaxval = 0;

if(goal1 > gmaxval)
 {
  gmaxval = goal1;
  str_cpy(goal_string,"GOAL1");
 }
if(gmaxval < goal2)
 {
  gmaxval = goal2;
  str_cpy(goal_string,"GOAL2");
 }

if(gmaxval < goal3)
 {
  gmaxval = goal3;
  str_cpy(goal_string,"GOAL3");
 }
if(gmaxval < goal4)
 {
  gmaxval = goal4;
  str_cpy(goal_string,"GOAL4");
 }
if(gmaxval < goal5)
 {
  gmaxval = goal5;
  str_cpy(goal_string,"GOAL5");
 }

if(str_cmpi(goal_string,"GOAL1"))
 {
  str_cpy(print_goal,"User seems to be interested in building Viilage");
 }
if(str_cmpi(goal_string,"GOAL2"))
 {
  str_cpy(print_goal,"User seems to be interested in kiiing enemies");
 }
if(str_cmpi(goal_string,"GOAL3"))
 {
  str_cpy(print_goal,"User seems to be a warrior");
 }
if(str_cmpi(goal_string,"GOAL4"))
 {
  str_cpy(print_goal,"User seems to explore the world");
 }
if(str_cmpi(goal_string,"GOAL5"))
 {
  str_cpy(print_goal,"User seems to complete the quest");
 }
print(print_goal);
waitt(100);
my_text.visible = off;
```

```
}

var rbgoal = 0;
function rulebase()
{
order_arrange();
analysis();
if(eventcounter > 2)
 {

  eventcounter = 0;

  rulebase_recent();
  rulebase_history();
 }
goal1 = max(goal1,0);
goal2 = max(goal2,0);
goal3 = max(goal3,0);
goal4 = max(goal4,0);
goal5 = max(goal5,0);
str_cpy(print_goal,"NO Information");
gmaxval = 0;
if(goal1 > gmaxval)
 {
  gmaxval = goal1;
  str_cpy(goal_string,"GOAL1");
  rbgoal = 1;
 }
if(gmaxval < goal2)
 {
  gmaxval = goal2;
  str_cpy(goal_string,"GOAL2");
  rbgoal = 2;
 }
if(gmaxval < goal3)
 {
  gmaxval = goal3;
  str_cpy(goal_string,"GOAL3");
  rbgoal = 3;
 }
if(gmaxval < goal4)
 {
  gmaxval = goal4;
  str_cpy(goal_string,"GOAL4");
  rbgoal = 4;
 }
if(gmaxval < goal5)
 {
  gmaxval = goal5;
  str_cpy(goal_string,"GOAL5");
  rbgoal = 5;
 }

print(goal_string);
waitt(40);

if(rbgoal == 1)
 {
  str_cpy(print_goal,"11User seems to be interested in building Viilage");
 }
if(rbgoal == 2)
 {
  str_cpy(print_goal,"22User seems to be interested in kiiing enemies");
 }
if(rbgoal == 3)
```

```
 {
  str_cpy(print_goal,"33User seems to be a warrior");
 }
if(rbgoal == 4)
 {
  str_cpy(print_goal,"44User seems to explore the world");
 }
if(rbgoal == 5)
 {
  str_cpy(print_goal,"55User seems to complete the quest");
 }
if(orderloc[19] < townbuilt_time)
 {
  str_cpy(print_goal,"User seems to be interested in Town Building and built it");
 }
if(orderloc[19] < smonsterdead_time)
 {
  str_cpy(print_goal,"User seems to be interested in complete quest and killed the Scorpion monster");
 }
if((orderloc[19] < knighthood_time) && (prki_val == 0))
 {
  str_cpy(print_goal,"User seems to be interested in complete quest and become a knight");
 }
if(orderloc[19] < fortcaptured_time)
 {
  str_cpy(print_goal,"User seems to be interested in capturing the fort and did it.");
 }



print(print_goal);
waitt(100);
my_text.visible = off;
}

string myst1;
string st1[90];
string myst2;
string st2[90];
string myst3;
string st3[90];
string myst4;
string st4[90];
string myst5;
string st5[90];

function rb1()
{
if(eventcounter > 2)
 {
  eventcounter = 0;
  order_arrange();
  rulebase_recent();
  rulebase_history();
 }
str_for_num(myst1,goal1);
str_cpy(st1,"G1-> ");
str_cat(st1,myst1);
str_for_num(myst2,goal2);
str_cpy(st2," G2-> ");
str_cat(st2,myst2);
str_for_num(myst3,goal3);
str_cpy(st3," G3-> ");
str_cat(st3,myst3);
```

```
str_for_num(myst4,goal4);
str_cpy(st4," G4-> ");
str_cat(st4,myst4);
str_for_num(myst5,goal5);
str_cpy(st5," G5-> ");
str_cat(st5,myst5);

str_cat(st1,st2);
str_cat(st1,st3);
str_cat(st1,st4);
str_cat(st1,st5);
print(st1);
waitt(100);
my_text.visible = off;

}
on_1 = rb1();
on_r = rulebase();
```

REFERENCES

1. Tom Brikowski, "*Virtual Reality in the Sciences".* http://www.utdallas.edu/~brikowi/Publications/haysTalk/node1.html

2. Jim W. Lai, *User-Modeling in Artificial Intelligence and Human-Computer Interaction.* http://www.io.com/~jwtlai/usermodel.html

3. Simon Goss and Clint Heinze, "*Recognizing User Intentions in Virtual Environment".* Proceedings of the Simulation Technology and Training Conference (SimTecT), Melbourne. March pp 247- 254 (1999).

4. Alicia Diaz and Ronald Melster, "*Patterns for Modeling Behavior in Virtual Environment Applications".* Second Workshop onHypermedia Development: design patterns in hypermedia (inconjunction with Hypertext 99). Darmstadt, Germany,February 1999.[2]

5. Jan Dijkstra, Harry J.P.Timmermans and Bauke D. Vries, "*Virtual Reality-based Simulation of User Behavior within the Built Environment to Support the Early Stages of Building Design".* In: Proceedings of The Eurographics Conference, Manchester, United Kingdom, September 4 – 7, 2001

6. Per Christiansson, Kjeld Svidt, Jens O. Skjaerrbaek and Rene Aaholm, "*User Requirements Modeling in Design of Collaborative Virtual Reality Design Systems".* CIB W78 - Information Technology in Construction - Mpumalanga, South Africa, 30th May - 1st June 2001

7. Richard A. Bartle, "*The Future of Virtual Reality".* http://www.mud.co.uk/richard/vrfuture.htm

8. Joseph Bates, "*Virtual Reality, Art, and Entertainment*". Presence: The Journal of Teleoperators and Virtual Environments – The MIT Press

9. Erik T. Mueller, "*Daydreaming in humans and machines: a computer model of the stream of thought*". Publisher: Cromland Inc.; (February 1990) ISBN: 0893915629