2010

Mythic game project addition of artificial intelligence and quest system components

Christopher Alan Ballinger

MYTHIC GAME PROJECT ADDITION OF ARTIFICIAL INTELLIGENCE AND QUEST SYSTEM COMPONENTS

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

by

Christopher Alan Ballinger

June 2010

MYTHIC GAME PROJECT ADDITION OF ARTIFICIAL INTELLIGENCE AND QUEST SYSTEM COMPONENTS

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Christopher Alan Ballinger

June 2010

Approved by:

[redacted]

David A. Turner, Chair, Computer Science — 5/3/2010 Date

[redacted]

Ernesto Gomez

[redacted]

Arturo I. Concepcion

ABSTRACT

In this project, we describe the design decisions and principles behind the artificial intelligence (AI) for a multiplayer online role playing game, and our use of an expert system to implement it. We explain how we organize AI rules into files, how those rules are assembled from a database, how AI is assigned to entities, the different types of AI, the different phases of AI, and how we manage facts used by AI. We also review some of the history behind the Mythic project, where it is headed, what an expert system is and why we chose to use one for our project. The result of our project is a design that allows us to have diverse AI behavior and flexibility to reuse code to create new behaviors, but may prove to be inefficient if implemented on systems with many players or many instances of AI running.

ACKNOWLEDGEMENTS

I would like to thank all of the other students that have been and are a part of this project, without their contributions my masters project would not have been possible. I would also like to thank all of my professors who have given me advice and encouraged me over the years and on this project, I would not have been able to come this far without their support. Finally, I would like to thank my family, for encouraging me and being there for me in my times of need. This project is also supported by NSF Award CNS-0938964.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Project Overview

Mythic is an active project focused on the development of all aspects of a multiplayer online role playing game (MORPG). The goal of Mythic is to develop a system with the capacity to offer similar capabilities as current MORPG games available on the market, as well as providing new innovative features to distinguish it from other MORPG games and make it attractive to potential players. The Mythic project has had a multitude of contributions from many different students, and has gone through several iterations since its inception. However, this project is focused on the artificial intelligence (AI) design and progress from the summer 2009 iteration of the Mythic project that was a part of a computer science masters project done by Chris Ballinger in the Department of Computer Science and Engineering at California State University, San Bernardino. The purpose of this project is to explain why we selected the technology we decided to use, describe how we designed the AI, and why we designed it that way, so that others interested in this field of study can learn from our experiences.

1.2 Client Overview

The client is written using Mircosoft Visual C++, and makes use of the ORGE graphics engine and Awesomium project. ORGE is used to display character models, object models, landscapes, and special effects such as spells. The Awesomium project allows us to embed a web browser anywhere inside the game and allow the user to interact with it. We use the Awesomium project to handle all of the graphic user interfaces(GUIs) such has dialog windows, login window, buttons and so on. When players interact with a GUI element, makes a non-persistent HTTP connection to the server and sends a message. When this message it received by the server, it will determine what javascript should be invoked in order to carry out the GUI request. The client also connects to the server through a persistent TCP connection. It uses this connection to send and receive messages related moving entities, clicking entities, removing entities, chat, and other events that are invoked or determined by entities other than the player.

1.3 Server Overview

The server is written is written in Java, and makes use of an Apache Tomcat web server, a MySQL database, and the Drools expert system. The Tomcat web server is what handles the persistent and non-persistent connection from the clients. The MySQL database is used for storing up to date data about a players character, and for storing data on all of the different entities and AI that exist in the game. The Drools expert system is used for carrying out rules for AI, and will be covered in detail later in this document.

1.4 Computational Thinking

This is the first year that the Mythic project has received support from the National Science Foundation for a project to promote computational thinking through community-based video game development projects. MORPG games are something many students in universities and the K-12 grades find interesting and enjoy playing. Involvement in community-based development projects such as Mythic may spark the interest of students in the creation of such games and open up an opportunity to get them to exercise their computational thinking skills while having fun. Computational thinking is a very important skill in today's world and can be applied to many fields of study, but it currently has no placement in the current K-12 curriculum[4]. However, by exposing students to computational thinking at an early age they will not only be more successful by developing this skill, but will also gain a better understanding of what computer science is and all of its fields[11]. Through video game development projects, the level of computational thinking students are exposed to could be scaled according to their grade. Freshmen students could focus on the process and logic of developing simple AI, while other students in the major could learn how to implement them.

1.5 Organization of this Document

Subsequent chapters in this document focus on the components and design of the AI. Chapter Two gives a brief overview of the history and current status of the project. Chapter Three gives an overview of what an expert system is and how it works, and why we decided to use an expert system to implement the AI. Chapter Four gives details on the requirements our the AI needed to fill, how we designed the AI to meet those requirements, how the rules are structured in files and in the database, and how information used by the rules are stored. Chapter Five gives a description of different types of AI we implemented and the rules that determine their behavior. Chapter Six describes how we tested the different AIs described in chapter five, and the results of those tests. Chapter Seven explains how to setup the code for the client and sever, and how to install other necessary compensates for those who wish to continue developing this project. Chapter Eight concludes with the advantages and disadvantages of our AI design, and future work to be done.

1.6 Definitions, Acronyms, and Abbreviations

The definitions, acronyms, and abbreviations used in the document are described in this section.

- AI: Artificial Intelligence
- CSUSB: California State University San Bernardino
- ES: Expert System
- GUI: Graphic User Interface
- Health: An integer value that represents how much damage a player or NPC can sustain. If this number reaches 0, the player or NPC is dead.
- Item: Represents a substance a player or NPC can utilize, at the cost of having the item deleted from their inventory.
- Inventory: A collection of items that a player or NPC possesses.
- Mana: An integer value that a player or NPC can consume in order to use a spell.
- MORPG: Multiplayer Online Role Playing Game
- NPC: Non-Player Character
- Spell: An action a player or NPC can select at the expense of some of their mana.

2. MYTHIC MULTIPLAYER ONLINE ROLE PLAYING GAME

2.1 History

Development on the Mythic project began in 2007, and has undergone a few rewrites since that time. The two biggest changes between these iterations were the programming languages used to develop the game (varying between Java, C++, and C#) and the graphics engine (varying between Horde3D[3], OGRE[10], and two different engines developed by students at CSUSB). We considered Horde3D for our graphic engine because it is a free, open source solution with impressive capabilities and a growing community. We considered OGRE because, like Horde3D, it is a free and open source solution with a large community, and has been used to produce the commercial game Torchlight. During this early development period, the storyline and game play mechanics started to become better defined as well.

2.2 Current Status

Currently, the Mythic project is transitioning from the OGRE graphics engine to the Unreal engine[5] for both our graphics and physics needs. We shifted to the Unreal engine because we felt that while OGRE and Horde3D were powerful tools, they were difficult to use and do not come with content creation tools. Furthermore, while the two engines developed at CSUSB showed a lot of promise, they were in a stage of development that was too early to be used. We feel that the Unreal engine will allow us to produce a game with modern graphics with with greater ease.

While there will be much for us to redo in the new iteration, the Unreal engine gives us access to many resources that allows us to set up a small online game out-of-the-box. However, these resources were not meant to be used for an MORPG style game, so our short-term goal is to modify the code to allow for an MORPG style game, with the expectation that when we accomplish this we will be able to get a basic game running. In the long-term, after we have a basic setup running, we hope to integrate some of our old resources, such as the AI, to this new project, and continue to develop features not yet fully implemented such as instanced battlefields, combat system and player interactions.

3. EXPERT SYSTEMS

3.1 Overview

3.1.1 Expert Systems

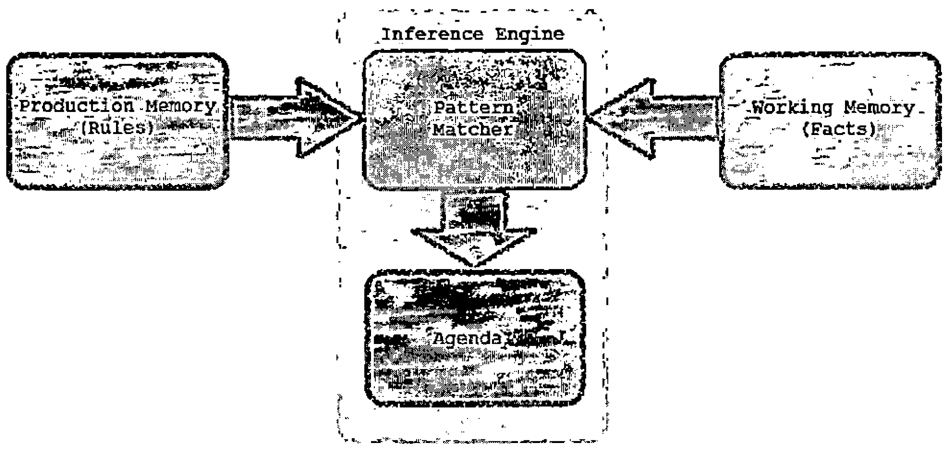


Fig. 3.1: Expert System Components

An Expert System (ES) is comprised of an inference engine, production memory and working memory, as shown in Figure 3.1. The production memory is where all the rules are stored, while the working memory is where all the facts are stored. A rule is a collection of conditions, and a collection of statements called the consequences(See Table 3.1), while a fact is a piece of data that may be used in the evaluation of a rule.

Tab. 3.1: Rule Structure

```
rule "name"
    <attributes>
    when
        <conditions>
    then
        <statements>
end
```

Inference Engine

An inference engine can be broken into two parts: the pattern matcher and the agenda. Whenever a fact is added to or removed from the working memory, the pattern matcher checks all of the rules in the production memory whose evaluation may change because of that fact. If all of a rule's conditions are met by the facts in working memory, it is placed into the agenda. If one or more conditions of a rule previously added to the agenda are no longer met, it is removed from the agenda. When all rules have been evaluated and no more changes to the working memory have been made, the consequences of a rule on the agenda will be executed. If this rule's consequences modify the working memory, then the rules in the production memory are checked again and the rules on the agenda are updated. This cycle continues until there are no more rules on the agenda, as illustrated by Figure 3.2.

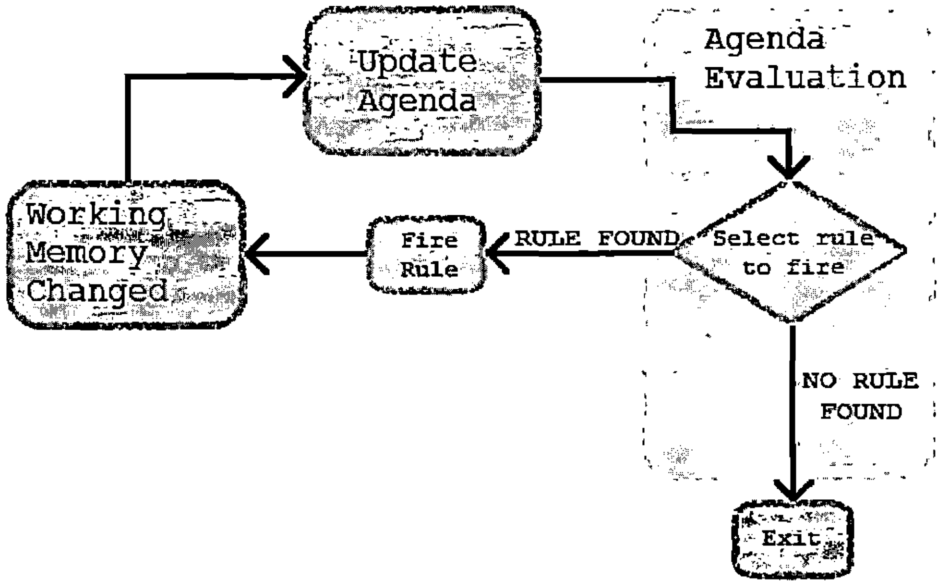


Fig. 3.2: Agenda Cycle

3.2 Reasons for Using an Expert System

We believe that an expert system provides us with the best option for creating AI. Unlike a finite state machine, which may cause behavior a player can easily predict and would require a massive amount of states[9], an expert system would make it easier to create behavior that is less predictable. This is because the expert system selects the action(s) to take based on the facts given to it, and the previous decision or action it took does not have factor into what it selects to do next. It also makes the rules easier to write, since figuring out an order in which they must occur is not necessary at every step. Expert systems have been used to make AI for games before, but usually they are for board games ranging in difficulty from tic-tac-toe[8] to chess[1]. Some more modern games may be making use of expert systems with success as well, such as a previously unnamed game using a specially designed expert system[7], but none seem to use commercially available expert systems such as the one we used for our project. The Soar Quakebot (which uses the Soar architecture to make AI for NPCs in the game Quake II) has also experimented with using an expert system to do NPC AI in a video game, and while they report that the behavior from the NPCs is intelligent and responsive[6], it took more than 800 rules in order for them to do so[9].

3.3 Drools

We selected Drools[2] as the expert system in our project. The reason we selected Drools over other expert systems such as CLIPS and JESS is primarily because it was a free, open source solution and works seamlessly with our Java code, but also because it is very well supported and has a very active community. The pattern matching algorithm Drools uses is a modified Rete algorithm called ReteOO, which is optimized for object-oriented systems. The reason we decided to use Drools for the AI portion of the project is because Drools is a proven technology in other fields, and we wish to see if its performance would be suitable for our MORPG. We also saw the potential flexibility that it would give our system and believed that to be an important asset.

3.3.1 How Drools is Used

Currently we only use Drools for deciding what actions a non-player character (NPC) should take, what dialog they should generate, and for the effects of spells and abilities (something a player or NPC can select to perform a special attack or to defend from an attack). All entities in the game (any object the player can click or interact with) are capable of having three AI sessions attached to them, and each of these sessions are created from a knowledge agent.

Sessions

A session in Drools is an instance of production memory, working memory and inference engine. An instance of a session is created from a knowledge agent as shown in Figure 3.3. Instances of sessions created from the same knowledge agent are totally independent of each other, changes to the working memory or agenda of one session do not affect the working memory or agenda of another session. This allows us to have multiple NPC's with the same AI from the creation of one knowledge agent.

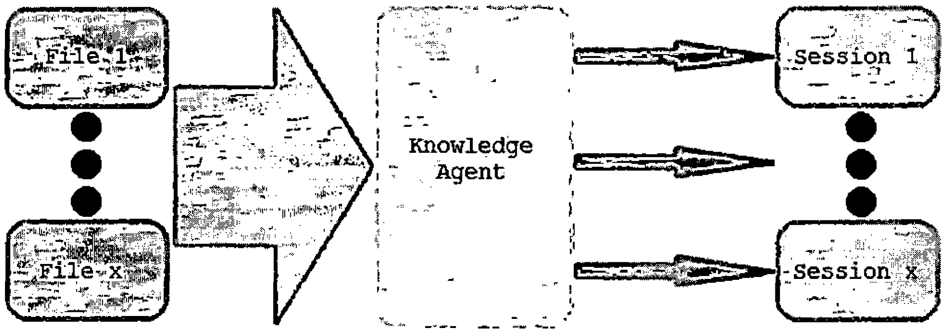


Fig. 3.3: Knowledge Agent and Session Creation Process

Knowledge Agents

A knowledge agent is basically a collection of rules that is used to create a session, shown in Figure 3.3. All knowledge agents are created once when the system first starts, and entities create their sessions from the agents after that. The creation of the knowledge agent is a expensive process, while creating an instance of a session from it is fairly lightweight. A knowledge agent is created from one or more rule files that contain various rules. When multiple files are used to create a knowledge agent, the agent will treat all of the rules as if they were contained in one file. All knowledge agents are also configured to check the files they are comprised of for changes every

30 seconds. If any changes are detected, the knowledge agent will rebuilt itself with the updated files, allowing for changes to AI behaviors on-the-fly.

4. ARTIFICIAL INTELLIGENCE DESIGN

4.1 Artificial Intelligence Design Requirements

The setting of Mythic is a fantasy medieval-type era, where people strive to gain a better understanding of magic instead of technology. There are three countries in this world that have different views on magic. The Kyrians are a people who rely solely on their physical strength, believing that using magic could lead to endangering the planet. The Sieric people are the opposite of the Kyrians, believing that one's value and position in life should be based on their magical prowess, and are always striving to increase their magic abilities. The Nochi believe in a middle ground, that using magic in moderation is safe, but using it to break the natural order of things could be a great danger, thus magic use should be strictly monitored. In addition to the conflicts between these three cultures, another threat to all three nations begins to stir, ancient god-like beings known as the Mythic. These beings take on nightmarish figures and attack people of all three lands. It is up to the player which country they wish to ally themselves with, and help defend it from the other nations while trying to learn what the Mythic are and how to stop them. In a world set in constant turmoil and battles, we need to design AI for NPCs so they know how to successfully defeat foes, assist allies, and react to players in accordance with how their culture feels about the players culture. We also need AI for spells and abilities that are capable of performing an almost unlimited amount of different actions, as one would expect from magic.

4.2 Artificial Intelligence Types

There are three different types of AI that we use Drools for. The first type is the effects of spells and abilities. When a player or NPC decides to use a spell, the only decision that entity is making is what spell to use and the intended target. The spell itself has it's own AI that determines if the spell is successful, how effective it is, if any additional effects will take place, if other nearby entities other than the target are affected, etc. When a spell has successfully been cast, it's effects are carried out by adding new facts and/or modifying all the relevant facts for all entities that are affected by the spell.

The second AI type is for dialog NPCs. The AI for these NPCs simply consists of looking at various facts, and then assembles dialog appropriate for the character talking to them based on those facts. As an example, if a male character talks to a NPC, the NPC might respond "We don't need a man for this job." while a female character who talks to the same NPC might get the dialog "Ah, about time a strong woman showed up." Dialog NPCs are also capable of giving players quests to participate in (a series of tasks that the player must accomplish). This is done by storing a fact in the database for the player that signals what part of the quest the player is on, so NPCs involved with the quest will react to the player accordingly, as shown in Figure 4.1.

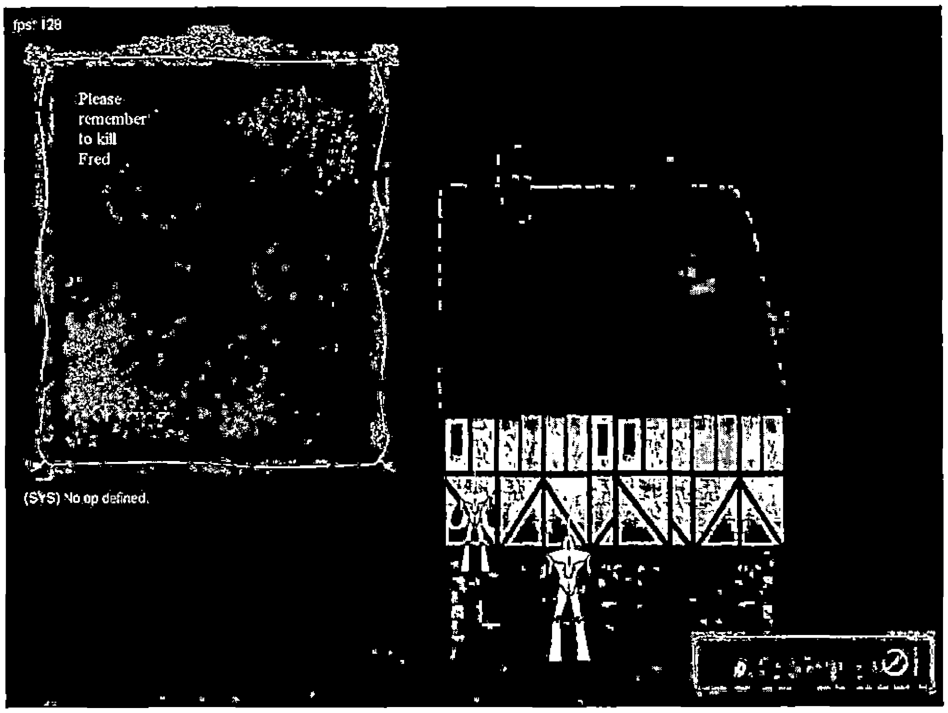


Fig. 4.1: A Dialog Non-Player Character's (left) Response to a Player(right) who has Previously Talked to him

The third type of AI is for combat NPCs, which require an AI that tells them how to to intelligently attack or defend themselves from a player or another NPC. Figure 4.2 shows two NPCs that have decided to pursue a player and attack him.

Fig. 4.2: Two Combat Non-Player Characters(left) Following and Attacking a Player(right)

4.3 Artificial Intelligence Sessions

As was mentioned before, every entity has three AI sessions attached to them. In light of this, and with one of our goals being to have maximum flexibility with the AI to create unique behaviors, there are no requirements to what type of AI these sessions perform. In our AI design we name these sessions the Status Check Session, Main Session and Activate Session. The status check session and main session are used during the update phase, with the status check session always being used first. The activate session is used whenever the entity is selected or right-clicked by a player. This process is illustrated by Figure 4.3. Any entity is capable of using any combination of these sessions, including all or none of them.

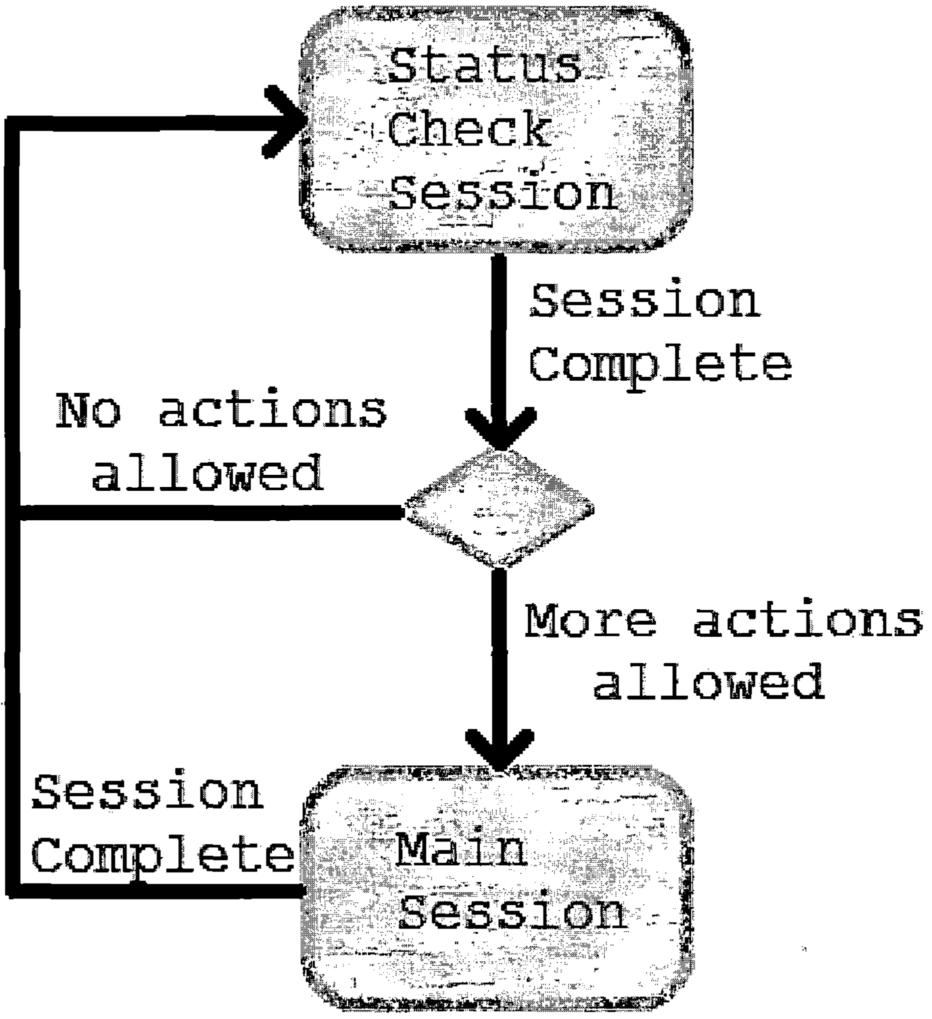


Fig. 4.3: Status Session and Main Session Flow

4.3.1 Status Check Session

In this session, the AI checks the status of the entity to see if a session for the main session has to be created or not. Typically in this session the entity checks if it is dead or alive, or to see if there are any residual effects that must be carried out from a spell that was previously performed on it that may affect its main phase. If it is determined that the NPC cannot take any further action, an instance of the main phase will not be created and will be skipped. Table 4.1 shows an example of a rule that is sometimes used in the status check session.

Tab. 4.1: Status Check Rule Example

```
rule "Death Check"
    when
    $self : Entity(facts[Fact.currentHealthKey]!=null,
                        facts[Fact.currentHealthKey] <= 0,
                        facts[Fact.respawnMilliKey]==null)
    then
        System.out.println("Death penality");
        $self.insertSessionPersistantFact(Fact.stopBeingLootableMilliKey,
                SimulationLoop.currentMillis + 5000);
        $self.insertSessionPersistantFact(Fact.respawnMilliKey,
                SimulationLoop.currentMillis + 10000);
        $self.setAllowUpdateAgent(false);
        $self.changeInteractAgent("dead_dialog");
end
```

4.3.2 Main Session

The purpose of the main session varies greatly depending on the type of NPC it is attached to, but it is most commonly used for combat NPCs. When dealing with combat NPCs, we split the rules into two separate phases: the Detection Phase and the Decision Phase.

Detection Phase

In this phase, the NPC may try to see if it currently has a target, and if so, check if that entity is still a valid target. If the NPC's previous target is no longer valid, or if it did not have a valid target, it will attempt to find the closest valid entity target within its detection area(shown in Table 4.2). Regardless of if the NPC has a target or not, when the Detection phase is over it moves onto the Decision phase.

Tab. 4.2: Detection Rule Example

```
rule "Detect New Target"
    agenda-group "detection"
    auto-focus true
    when
        $self : Entity(facts[Fact.targetKey] == null)
    then
        Float detectionDist =
            (Float) $self.getFacts().get(Fact.detectionDistKey);
        Entity nearestAvatar = null;
        Float nearestAvatarDist = 0f;
        for(Entity avatar : $self.getArea().getAvatars()){
            float dist = $self.distanceFrom(avatar);
            if((dist <= nearestAvatarDist) || (nearestAvatar == null)){
                nearestAvatar = avatar;
                nearestAvatarDist = dist;
            }
        }
        if((nearestAvatarDist < detectionDist) && (nearestAvatar != null)){
            $self.insertSessionPersistantFact(Fact.distToTargetKey,
                nearestAvatarDist);
            $self.insertSessionPersistantFact(Fact.targetKey, nearestAvatar);
            update($self);
        }
        drools.setFocus("decision");
end
```

Decision Phase

The decision phase will vary greatly between different combat NPCs, but generally most combat NPCs can do two things in this phase. If the NPC is not attacking or being attacked by another entity, it will patrol some area. Patrolling involves following a path, wandering randomly in an area(shown in Figure 4.4), or moving about the world according to some algorithm, until it finds something it can act on.

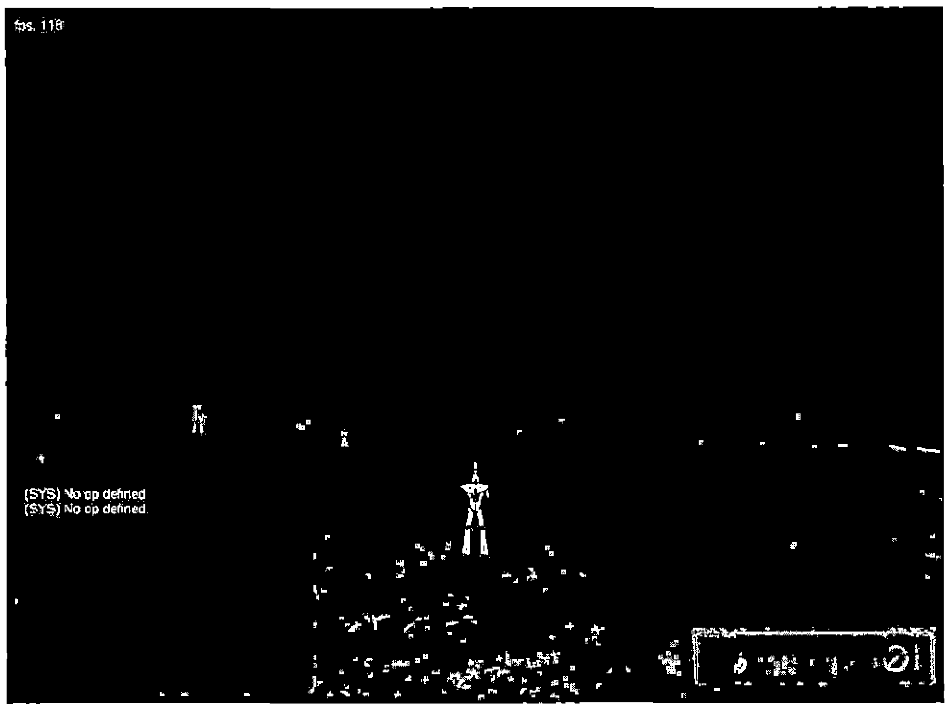

```
fps: 118
(SYS) No op defined
(SYS) No op defined
```

Fig. 4.4: Two Non-Player Characters(back) Randomly Selecting Points in an Area and Walking to them

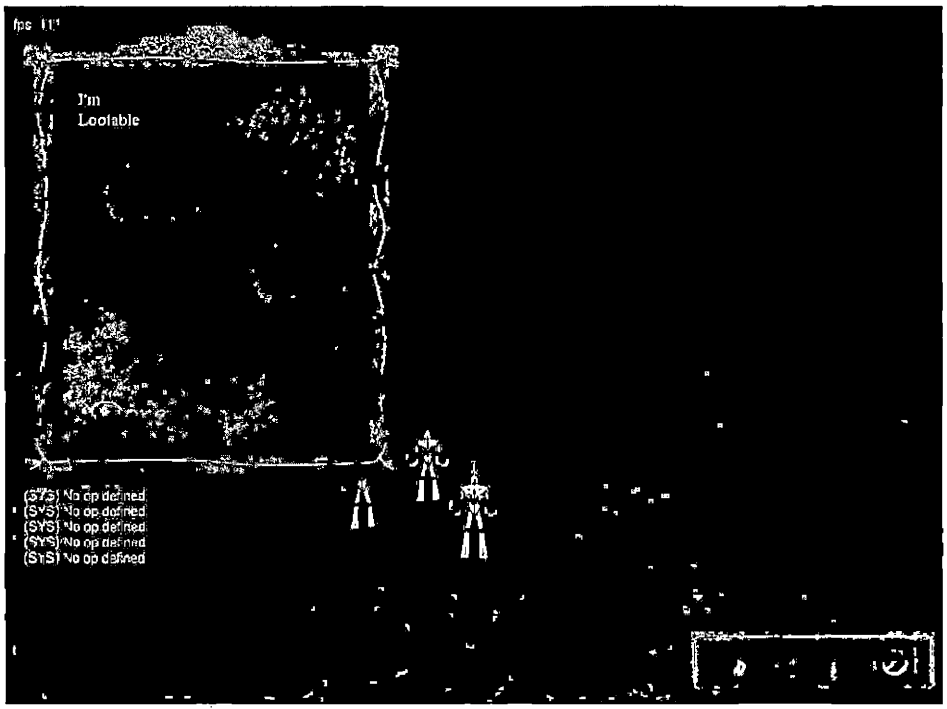
If the NPC is attacked by or detects an entity it can attack, it will try to decide the best course of action it should take, such as which item or spell to use(shown in Table 4.3), whether it should attack or retreat, etc. Other unique behavior could also be implemented in the Decision phase as well.

Tab. 4.3: Decision Rule Example

```
rule "Select Fireball"
    agenda-group "decision"
    when
        $self : Entity(
            facts[Fact.targetKey] != null,
            facts[Fact.selectedAbilityKey] == null,
            facts[Fact.currentManaKey] != null,
            facts[Fact.currentManaKey] >= 20,
            ( (facts["fireball"+Fact.earliestInvokeKeySuffix] == null) ||
                (facts["fireball"+Fact.earliestInvokeKeySuffix]
                    < SimulationLoop.currentMillis) )
            )
    then
        $self.insertSessionPersistantFact(Fact.selectedAbilityKey,
            "fireball");
        $self.insertSessionPersistantFact(Fact.selectedAbilityRangeKey,
            300f);
        $self.insertSessionPersistantFact(Fact.selectedAbilityCooldownKey,
            5000L);
        update($self);
end
```

4.3.3 Activate Session

This session is only created and activated when a player clicks on the NPC. The AI in this session also varies in purpose depending on the NPC it is attached to. For dialog NPCs, this is where the AI generates dialog and sends it to the player. For portals, this is where the AI determines if a player is allowed to move to another area or which area the portal will move the player to. For a combat NPC, this is where the AI decides if the player is performing an attack on it or is attempting to take items off of its dead body, as illustrated by Figure 4.5.



I'm
Lootable

```
(SYS) No op defined
(SYS) No op defined
(SYS) No op defined
(SYS) No op defined
(SYS) No op defined
```

Fig. 4.5: Right-Clicking on a Recently Killed Combat Non-Player Character Opens this Dialog Instead of Performing an Attack

4.4 Rule Structure

While the rules themselves do not have any strictly enforced structure they must follow, there is only one way to assign rules to a knowledge agent, or to assign a knowledge agent to a NPC. This section discusses how that assignment is done in the database and how the rules are organized into different files for maximum re-usability.

4.4.1 Database Structure

Assigning Rules to a Knowledge Agent

In order to create rules, two tables are required in the database: the agent name table(see Table 4.4 for insert examples) and the file assignment table(see Table 4.5 for insert examples). The agent name table has only one column, and contains the names of all valid agents. The file assignment table has two columns: the file name and the agent name. When the system starts up, it gets all the agent names from the agent table, it looks up all entries in the file assignment table that contain the same agent name, and creates the agent from all the files named in those entries. The same file can be assigned to more than one agent, allowing us to reuse AI rules and create many new unique behaviors without any extra work.

Tab. 4.4: Agent Name Table Insert Examples

```
insert into kbase(id) values ('basic_ai');
insert into kbase(id) values ('elf32');
```

Tab. 4.5: File Assignment Table Insert Examples

```
insert into drools_file(id,kbase_id) values('Detect.drl','basic_ai');
insert into drools_file(id,kbase_id) values('Patrol.drl','basic_ai');
insert into drools_file(id,kbase_id) values('Combat.drl','basic_ai');
insert into drools_file(id,kbase_id) values('Attacker.drl','basic_ai');
insert into drools_file(id,kbase_id) values('Patrol.drl','elf32');
```

Assigning a Knowledge Agent to a Non-Player Character

In order to assign rules to a NPC, there are three fields in the entity database, one for each session, where the name of the agent that is used for that session is placed(see Table 4.6 for insert examples). If a NPC does not use any AI for a session, the value null is passed in for that session.

Tab. 4.6: Assigning Artificial Intelligence to Non-Player Character Examples

```
insert into entity (id, mythic_type, home_area_id, home_x, home_z,  home_yaw,
    movement_speed, checkstatus_kbase_id, main_kbase_id,  activate_kbase_id)
    values ('fred', 'human', 'homeArea', 600, 600, 0, 0.02, 'fred_status',
    'basic_ai', 'take_damage');
insert into entity (id, mythic_type, home_area_id, home_x, home_z,  home_yaw,
    movement_speed, checkstatus_kbase_id,  main_kbase_id, activate_kbase_id)
    values ('ted', 'human', 'homeArea', 400, 400, 0, 0.02, 'check_status',
    'basic_ai','take_damage');
insert into entity (id, mythic_type, home_area_id, home_x, home_z,  home_yaw,
     movement_speed, checkstatus_kbase_id,  main_kbase_id, activate_kbase_id)
     values ('jed', 'human', 'homeArea', 200, 200, 0, 0.02, null, null,
     'jed_dialog');
```

4.4.2 Rule Organization

Items and Spells

Rule organization for item and spell effects are very simple, they only check to make sure they have a valid target, and then carry out the expected consequences of that rule, as shown in Table 4.7. There is generally only one spells/item effect per file.

Tab. 4.7: Item Rule Example

```
rule "Potion"
    when
        $self : Entity()
    then
        $self.getInventory().remove("Potion", 1);
        $self.getInventory().numberOfItemsOfType("Potion") + ".");
        Integer decreasedValue =
            (Integer) $self.getFacts().get(Fact.currentManaKey);
        int newHealth = ((Integer) $self.getFact(Fact.currentHealthKey)) + 2;
        $self.insertSessionPersistantFact(Fact.currentHealthKey,newHealth);
end
```

Dialog

The dialog rules are equally simple, containing a list of rules that generate dialog based on various facts, as shown in Table 4.8. However, in order for the NPC to form a coherent sentence, creating an order for some of the rules to be fired will be necessary. Some common greetings and such can be placed in separate files for reuse, but generally every NPC generates different dialog, and this dialog in contained in a single file for each NPC.

Tab. 4.8: Dialog Rule Example

```
rule "Jeds Dialog"
    lock-on-active true
    when
        $fact : Fact(key == Fact.activaterAvatarKey, $avatar : value)
    then
        System.out.println("Jed Dialog");
        Entity avatar = (Entity)$avatar;
        ArrayList<String> dialogPages =
            (ArrayList<String>)avatar.getFact(Fact.dialogPagesKey);
        dialogPages.add("Hello.");
        dialogPages.add("Goodbye.");
        update($fact);
end
```

Combat Non-Player Characters

The organization of the main phase for combat AI components is more complex. Rules belong to one of two agenda groups: detection or decision. An agenda group makes it so that only rules in the group that has focus are allowed to have their consequences fired. In the main phase for combat AI, the detection phase starts with the focus, and then hands the focus to the decision phase when it is finished, as can be seen in Table 4.2. Rules for detection, patrolling, and battle decisions are kept in separate AI files from each other to maximize our ability to reuse them.

While this is how we organized our rules for the project, this convention is not a requirement in order to create AI. However, AI that do not follow these conventions should not be mixed with AI files that do, as they may not be compatible.

4.5 Fact Management

4.5.1 Class Design

Every entity contains two hash maps: one named facts and another named init facts. When the system first starts, the init facts map is populated with facts stored in the database, such as how much damage the entity can take. The facts in the init facts map are also copied into the facts map. The facts map acts as the working memory for all the AI sessions that entity contains. The reason we do this is because if Drools manages its own working memory then all the facts would have to be reinserted every update phase in order to populate the agenda. So it is simpler to manage our own list of facts and directly use it as the working memory in a stateless session. Whenever an entity is ready to reappear in the game after being defeated, all of the contents in the facts map is deleted and the facts from the init facts map are copied over again, resetting it to it's default state. Whenever an old fact in the database is updated or a new fact is inserted into the database, we call a function that makes the database entry and updates the entities init facts map as well.

4.5.2 Database Design

The fact table has four columns: the fact key, the fact value, the value type and the entity id. The fact key and fact value are the pair that are inserted into the fact hash tables, the value type is a single character that identifies the type of the fact value ('i' for integer, 'f' for float, or 's' for string), and the entity id identifies the entity to whom the fact belongs to. Insert examples for this table are shown in Table 4.9.

Tab. 4.9: Fact Table Insert Examples

```
insert into fact(fact_key, fact_val, val_type, entity_id)
    values ('detectionDistance', '100', 'f', 'ted');
insert into fact(fact_key, fact_val, val_type, entity_id)
    values ('pursueDistance', '400', 'f', 'ted');
insert into fact(fact_key, fact_val, val_type, entity_id)
    values ('currentMana', '100', 'i', 'ted');
insert into fact(fact_key, fact_val, val_type, entity_id)
    values ('currentHealth', '3, 'f', 'ted');
```

5. IMPLEMENTATION

5.1 Dialog Artificial Intelligence

We implemented three different NPCs that make use of dialog AI. The first NPC gives players different dialog depending on the race(a human or an elf) of the player that clicked them. The second and third NPC involve giving players different dialog/events based on what part of a quest the player is on. The second NPC generates different dialog based on what phase of the quest the player is on(either hasn't started the quest, is currently assigned the quest, or has previously completed the quest). The third NPC is also a combat NPC, however it gives the player an item when it is defeated based on what phase of the quest they are on(if the player is currently assigned the quest they player receives a tooth, if the player is on any other phase of the quest or already has the tooth they receive nothing).

5.2 Combat Artificial Intelligence

We created two different NPCs named Fred and Ted that have combat AI. The Fred NPC is involved in a quest, and as a result has AI that is made up of more rules. The Ted NPCs AI is much simpler, and the rules that comprise its AI is a proper subset of Fred NPCs. With this being the case, the following subsections describe the AI for Fred NPC.

5.2.1 Status Check Session

For the status check session, we created an AI that has three rules: Death Check, Lootable Check, and Respawn Check. Death Check determines if the NPCs health has just reached zero, and if so it sets a time in the NPCs fact for when it stops being lootable and when it should respawn, and changes the activate session to an instance of the dead_fred_dialog agent. The Lootable Check rule determines if the time for being lootable has passed, and if so it removes the entity from the map so that the player can no longer interact with it, and removes the time for being lootable from the NPCs facts. The Respawn Check rule determines if the time to respawn has passed, and if so it removes the time to respawn from the NPCs facts, adds the entity back into the world with its default facts, and sets the activate session back to an instance of the take_damage agent. See Table 5.1 for the pseudo code for this session.

Tab. 5.1: Status Check Session Pseudo Code

```
rule "Death Check"
    when
        Health <= 0
        RespawnTime == NULL
    then
        LootableTime = CurrentTime + 5000;
        RespawnTime = CurrentTime + 10000;
        ActivateSession = dead_fred_dialog;
end
rule "Lootable check"
    when
        LootableTime <= CurrentTime
    then
        LootableTime = NULL;
        HideBody();
end
rule "Respawn check"
    when
        RespawnTime  <= CurrentTime
    then
        LoadDefaultFacts();
        ActivateSession = take_damage;
        RespawnTime = NULL;
        ShowBody();
end
```

5.2.2 Activate Session

We created two different agents that are used at different times for this session. The default agent used is the take_damage agent. This agent is created with only one rule named "Take 1 damage" that decreases the NPCs health by one whenever it is clicked, as shown in the pseudo code in Table 5.2.

Tab. 5.2: Take Damage - Activate Session Pseudo Code

```
rule "Take 1 damage"
    when
        Health > 0
    then
        Health--;
end
```

However, at certain times the status check session will change the activate session to an instance of the dead_fred_dialog agent. This agent also contains one rule named Avatar Finds, Tooth, which checks the quest status and inventory of the player that clicks it. If they player is not on the correct quest or already has the item Fred's Tooth in their inventory, they will receive a message that they find nothing. Otherwise, they will receive a message that they found the item and have it placed in the first available slot in their inventory. Pseudo Code for this process is shown in Table 5.3.

Tab. 5.3: Avatar Finds Tooth - Activate Session Pseudo Code

```
rule "Avatar Finds Tooth"
    when
        PlayerWhoClicked != NULL
    then
        if( PlayerWhoClicked.Quest(FredsQuest) != Started ||
            PlayerWhoClicked.InventoryContains("Freds Tooth"))
        {
            SendDialog(You Find Nothing);
        }
        else
        {
            SendDialog(You find Freds Tooth);
             PlayerWhoClicked.GiveItem(Freds Tooth);
        }
end
```

5.2.3 *Main Session*

The main session for the combat AI is created from multiple files. The rules for the detection phase of the main session come from one file, but the rules for the decision phase come from multiple files. We created one implementation of the main session that is used by two combat NPCs.

Detection Phase

The detection phase contains two rules, the Detect New Target rule and the Validate Target rule. In the Validate Target, if the NPC does have a target, but the target has moved out of its range, then the NPC sets its current target to null. The Detect New Target rule determines if the NPC target is set to null(meaning it does not have a target), and if so it will look for the closest player within its range. If it finds such a payer it will set that player as its active target. After all rules in the detection phase have been fired, rules in the decision phase are allowed to fire next. Pseudo code for these rules are shown in Table 5.4.

Tab. 5.4: Detect Phase - Main Session Pseudo Code

```
rule "Detect New Target"
    agenda-group "detection"
    auto-focus true
    when
        CurrentTarget == NULL
    then
        Player PossibleTarget = ClosestPlayer();
        if(PossibleTarget.Distantance < DetectionDistance)
            CurrentTarget = PossibleTarget;
        setFocus("decision");
end

rule "Validate Target"
    agenda-group "detection"
    auto-focus true
    when
        CurrentTarget != NULL
    then
        if(CurrentTarget.Distance >= PursueDistance)
            CurrentTarget == NULL;
        else
            setFocus("decision");
end
```

Decision Phase

The decision phase is comprised of four rules: Select new destination, Ability Range, Select Fireball and Select Melee. The Select new destination rule determines if the NPC does not have a target and if it has reached its previous selected point. If both these conditions are true, then the NPC will select a random point in a radius, and will proceed to walk in a straight line to that point. If the NPC does have a target, then it will decide what ability it should use. The NPC has two abilities it can select from, for which the conditions for selecting are defined by the Select Fireball and Select Melee rules. In order for the Select Melee rule to be selected, it must not have been selected before in the last 3 seconds, the NPC must be very close to the player to use and no other ability must be currently selected. If this rule is selected then three facts are added to the NPC. The selectedAbility fact that contains the string Melee, the selectedAbilityRange fact that contains the distance 20, which the NPC must be within to use the ability, and the selectedAbilityCooldown fact that contains 3000ms, which is how long the NPC must wait before it can use the melee ability again. The Select Fireball is the same as the Select Melee rule, except in order to be selected the NPC must have at least 1 mana and it must be at least 5 seconds since the ability was last select. The consequences of the rule similar to Select Melee as well, with the only difference being that selectedAbility is set to Fireball, selectedAbilityRange is set to 300, and selectedAbilityCooldown is set to 5000ms. The "Ability Range" rule is selected when the selectedAbility fact is present and the NPC has a target. If the player is out of the range of the NPCs selected attack, the NPC will run towards the player until they are in range. Once the NPC is in the correct range to use the selected ability it activates the AI for the corresponding ability, and removes the selectedAbility facts from the NPC. This process is shown in the pseudo code in Table 5.5 and Table 5.6.

Tab. 5.5: Decision Phase - Main Session Pseudo Code

```
rule "Select new destination"
    agenda-group "decision"
    when
        CurrentPosition != TargetPosition
        CurrentTarget == NULL;
    then
        TargetPosition = RandomPosition();
end
rule "Ability Range"
    agenda-group "decision"
    lock-on-active true
    when
        SelectedAbility != NULL
        CurrentTarget != NULL
    then
        if(CurrentTarget,Range > SelectedAbility.Range)
            TargetPosition = CurrentTarget.Position();
        else
        {
            SelectedAbility.Activate();
            SelectedAbility == NULL;
        }
end
```

Tab. 5.6: Decision Phase Available Abilities- Main Session Pseudo Code

```
rule "Select Melee"
    agenda-group "decision"
    when
        CurrentTarget != NULL
        SelectedAbility == NULL
        MeleeTimer <= CurrentTime
    then
        SelectedAbility = Melee;
end
rule "Select Fireball"
    agenda-group "decision"
    when
        CurrentTarget != NULL
        SelectedAbility == NULL
        FireballTimer <= CurrentTime
    then
        SelectedAbility = Fireball;
end
```

5.3 Item and Spell Artificial Intelligence

We implemented three different AI to be used for spells or items. The first AI is used for Potion items. A potion can be used by either a player or a NPC, and raises the users current health by five. The pseudo code for this is shown in Table 5.7.

Tab. 5.7: Potion Pseudo Code

```
rule "Potion"
    when
        EntityUsing != NULL
    then
        EntityUsing.Inventory().Remove(Potion);
        EntityUsing.Health += 2;
end
```

The second AI is used for Fireball spells. When a player or NPC uses a fireball spell, it causes the target to lose 1 health and the user to lose 1 mana. The pseudo code for this is shown in Table 5.8.

Tab. 5.8: Fireball Pseudo Code

```
rule "Fireball"
    when
        EntityUsing != NULL
    then
        EntityUsing .Mana -= 1;
        EntityUsing.CurrentTarget.Health -= 1;
end
```

The third AI is used for Melee attacks. When a player or NPC uses a melee attack, it only causes the target to lose 1 health. The pseudo code for this is shown in Table 5.9.

Tab. 5.9: Melee Pseudo Code

```
rule "Melee"
    when
        EntityUsing != NULL
    then
        EntityUsing.CurrentTarget.Health -= 1;
end
```

Every current spell and item AI is made up of only one rule. The condition for the rule of any spell or item is always the same, and that for just that it is invoked by an entity. This is because the conditions for using the ability are determined by other rules, such as the ones in the main session of the combat AI. Item and Spell AI are only used for carrying out the effects of the spell.

6. TESTING

6.1 Dialog Artificial Intelligence

To test the simple dialog NPC, we created one human character and one elf character and had each of them take turns talking to the NPC, with the correct dialog being generated every time for each character. To test the two NPCs involved with a quest, we performed two tests. In the first test, we had a character try to kill the combat NPC several times, then talk to the dialog NPC several times, then kill the combat NPC several times again, and finally talk to the dialog NPC several times again. For the second test we repeated the same steps as in the first test, but had the character log out and log back in after each step to show that the quest phase and item were being stored and retrieved properly from the database. In each of these tests, the correct response was given every time by both NPCs.

6.2 Combat Artificial Intelligence

6.2.1 Activate Session

To make sure that the AI in the activate session was working correctly, we had a player rapidly click on this NPC and timed how long it would give the player a dialog and how long it would generate no dialog to make sure that these phases were lasting for the correct duration we had set.

6.2.2 Main Session

Detection Phase

The decision phase was tested by having a player slowly approach one of the combat NPCs until it detected the player. Once the NPC had detected the player, the player would run a certain distance and stop to allow the NPC to catch up. The player would repeat this, increasing the distance he would run each time, until the NPC would lose the player as a target. We repeated this test several times. The distance that the NPC would detect the player and lose the player as a target was approximately the same each time. This is previously shown in Figure 4.2.

Decision Phase

To test the behavior of the combat NPCs who do not have a target, we had a player observe the combat NPCs for a few minutes while staying out of their detection range. We then allowed a combat NPC to detect the player, then lose the player as a target and had the player obverse the NPCs movements. In this case, when the NPC loses the player as a target, it selects a random point within its normal radius and walks in a straight line until it reaches that point regardless of distance. Upon reaching this point, it would resume selecting new points and walking to them as usual. This was previously shown in Figure 4.4.

To test the behavior the the NPCs when they have a target, we had a player allow a NPC to detect them as a target. For the first few minutes, if the player moved out of the NPCs melee distance, the NPC would stay back a moment to perform a fireball attack and then walk towards the player to perform a melee attack. However, once the NPC ran out of mana(rendering it unable to use the fireball attack anymore), the NPC would start to move closer to the player much sooner. This was previously shown in Figure 4.2.

6.2.3 *Status Check Session*

These two events are tested during the main phase and activate phase, since these events are a required for the main phase and activate phase to function properly. However, we also tested the NPC after disabling different parts of the AI in the status check session to show that the main session and activate session are dependent on the status check session. This was previously shown in Figure 4.5.

6.3 *Item and Spell Artificial Intelligence*

The melee and fireball attacks are tested as part of the combat AI testing. To test the potion AI, we place several potions in a players inventory. Whenever the player uses a potion, it is removed from their inventory and its effects are carried out. We had the player test the potions in a few different situations, such as during combat or while walking.

7. MAINTENANCE

The Mythic server can run under Windows, Mac and Linux. The Mythic client currently only runs under windows, but is designed to port easily to Mac and Linux.

Note: In the following, the symbol {tomcat} represents the path to the tomcat folder on your system and {workspace} refers to the path to your eclipse workspace; you need to replace these symbols with the actual pathnames they represent.

7.1 Server Setup

7.1.1 Prerequisites

- JDK 1.6 (Recommended: JDK 6 Update 14)
- Recent version of Eclipse IDE for Java EE Developers
- Download the zip file containing Tomcat; don't install as a service; instead, unpack the zip file somewhere in your file system
- MySQL (when configuring on Windows, check the box to add mysql to the bin path and set the admin password to admin)
- Get the most recent stable version of MySQL Connector/J and put into {tomcat}/lib
- Install Drools 5.0 Eclipse 3.4 Workbench into Eclipse
  (update site= http://downloads.jboss.com/drools/updatesite3.4/)
- Download Drools 5.0 binaries at http://jboss.org/drools/downloads.html

7.1.2 Setup

- Copy the server project folder on the CD to the desired folder.
- Create a library variable that points to {tomcat}/lib/servlet-api.jar. To do this, do the following.
  - Open Window ... Preferences.
  - Expand the Java node.
  - Expand the Build Path node.
  - Select the Classpath Variables node.
  - Click New and enter servlet-api for the name and click on File... and browse to and select {tomcat}/lib/servlet-api.jar.
  - Click through OKs to complete the definition.
- Create a database called mythic and create and populate its tables. To do this, start the mysql command line client as root. If you set the root password to 'root' then you would do this as follows (notice the absence of a space between -p and root):

  ```
  mysql -u root -proot
  ```

- At the mysql command line, run the following:

  ```
  mysql> create database mythic;
  ```

- Now exit mysql command line client and go back into eclipse. Run the ant build file database/build.xml. To do this, right click the file and select Run As ... Ant Build.

- Window ... Preferences ... Server Runtime Environments ... Apache Tomcat v6.0, check create new local server, then click Next. For the tomcat installation directory, browse to {tomcat}. Click Finish and OK.
- Window ... Preferences ... Drools ... Installed Drools Runtimes. Click Add..., Click a Create a new Drools Runtime... Navigate to wherever you want to store the Drools runtime. Click through to complete operation.

7.2 Client Setup

7.2.1 Software Prerequisites

- Visual Studio 2008

7.2.2 Setup

- Copy the client project folder on the CD to the desired folder.
- Inside Visual Studio, bring up the properties window for the mythic_client project
- In the Debugging window, set working directory to ..\Debug

8. CONCLUSION

8.1 Advantages

We have found that there are many advantages to our AI design. Our design gives us the ability to create new AI behavior and modify existing behavior on-the-fly, and allows us to mix and match previously created AI components to create new behavior with little extra effort. Our AI and NPC design also allows any NPC to be capable of any action; a treasure box that a player clicks on to collect items is the same type of NPC that is used for a combat NPC or a dialog NPC. The only difference between them is the AI they have been assigned. Even the character that a person controls is in essence an NPC without a main session attached to it. Our design also allows for AI to take control of the player's character, or for the player to take control of any NPC. This allows us to have maximum flexibility when it comes to creating a multitude of innovative and unique behaviors for NPCs to have. For instance, a treasure box could ask a riddle the player must solve in order to open it. If the player answers correctly, they can take an item out of it, but if answered incorrectly, it will take control of the player's character for a short period of time and force the player to defend as it attacks allies.

8.2 Disadvantages

While our approach works well for a small-scale game, it has yet to be tested on a system with many users or many NPCs. Further testing on a large-scale system may prove our solution to be an inefficient, since it may be too computationally intensive to implement for all entities in the game.

8.3 Future Work

Basic AI functions like combat, quests, items, spells, abilities and dialogs are essentially complete and are only lacking in variety. In the near future we hope to expand on these and add AI to other aspects of the game, such as deciding what items a merchant NPC has, and how much it sells them for based on player actions, as well as other factors.

When all AI aspects have been well developed, we hope to address the issue of efficiency. Depending on the severity of the problem, there are several possible solutions that may help. The best way to help alleviate the efficiency problem is probably though distributed computing. By using several servers that are each responsible for calculating the AI of a set of preassigned NPCs, efficiency may not become an issue. The issue could also be addressed through the game design, changing it so a much smaller number of NPCs exist in the game, but increasing the difficulty of the remaining NPCs.

We have also submitted a paper to the Foundation of Digital Games Conference based on the work done on the AI. We hope to make further developments in Mythic that we can submit to the conference in the future.

REFERENCES

[1] F. Hsu M. Campbell and J. Hoane. Deep blue system overview. In *ICS '95: Proceedings of the 9th international conference on Supercomputing*, pages 240–244, New York, NY, USA, 1995. ACM.

[2] JBoss Enterprise. Drools: Business logic integration platform. http://www.jboss.org/drools/.

[3] N. Schulz et al. Horde3d next-generation graphics engine. http://www.horde3d.org/home.html.

[4] G. Fletcher and J. Lu. Human computing skills: rethinking the k-12 experience. *Commun. ACM*, 52(2):23–25, 2009.

[5] Epic Games. Udk unreal development kit. http://www.udk.com/.

[6] J. Laird. It knows what you're going to do: adding anticipation to a quakebot. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 385–392, New York, NY, USA, 2001. ACM.

[7] J. Laird and D. Pottinger. Game ai: The state of the industry, part two. http://www.gamasutra.com/view/feature/3569/game_ai_the_state_of_the_.php, November 2000.

[8] R. Pilgrim. Tic-tac-toe: introducing expert systems to middle school students. In *SIGCSE '95: Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education*, pages 340–344, New York, NY, USA, 1995. ACM.

[9] S. Rabin. *Introduction to Game Development (Game Development)*. Charles River Media, Inc., 2005.

[10] Torus Knot Software. Orge open source 3d graphics engine. http://www.ogre3d.org/.

[11] J. Wing. Computational thinking. *Commun. ACM*, 49(3):33–35, 2006.