Mythic: Artificial Intelligence Design in a Multiplayer Online Role Playing Game

Chris A Ballinger
California State University San Bernardino, Dept. of Computer Science and Engineering
5500 University Parkway
San Bernardino 92407
1-909-537-5326
mtg15@hotmail.com

David A Turner
California State University San Bernardino, Dept. of Computer Science and Engineering
5500 University Parkway
San Bernardino 92407
1-909-537-5428
dturner@csusb.edu

Arturo I Concepcion
California State University San Bernardino, Dept. of Computer Science and Engineering
5500 University Parkway
San Bernardino 92407
1-909-537-5330
concep@csusb.edu

ABSTRACT

In this paper, we describe the design decisions and principles behind the 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.

Categories and Subject Descriptors
I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems

General Terms
Design

Keywords
Games, Artificial Intelligence, Expert Systems

1. INTRODUCTION

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 paper will focus 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 paper 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.

1.1 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[1]. 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[2]. Through video game development projects, the level of computational thinking students are exposed to could be scaled according to their grade. Younger students could focus on the process and logic of developing simple AI, while older students could learn how to implement them.

1.2 Organization of the paper

The first section of this paper will give a brief overview of the Mythic project's development history, current status, and future plans. The second section will explain what an expert system is, what expert system we selected, and how we used it. The third section describes the design of AI we developed, the phase design for the AI, how rules are represented in the database, how rules are organized in files, and how we manage facts used in evaluating rules. Finally, in our conclusion, we discuss advantages of our design, possible problems that may arise in the future, and our future plans.

2. MYTHIC MORPG

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[9], 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.

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 System (ES)

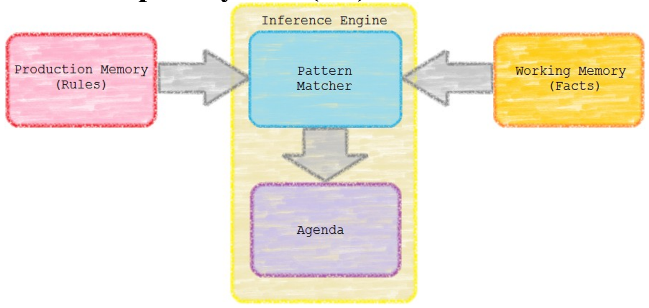


Figure 1: Expert System Components

An Expert System (ES) is comprised of an inference engine, production memory and working memory. 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”, while a fact is a piece of data that may be used in the evaluation of a rule.

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

Figure 2: Rule Structure

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.

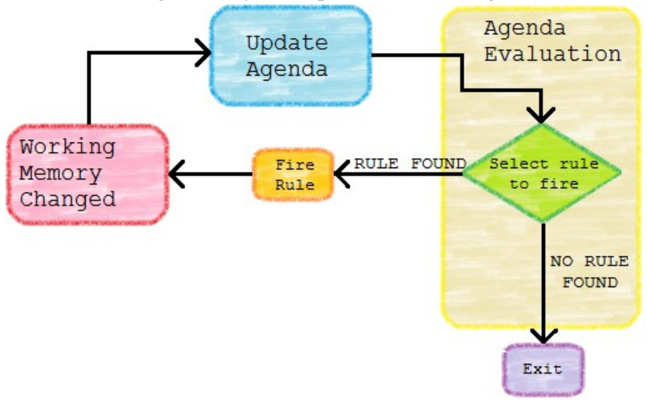


Figure 3: Agenda Cycle

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 state[8], 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[7] to chess[3]. 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[5], 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[8].

3.1 Drools

We selected Drools[6] 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. 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).

It's also worth noting that all entities in the game (any object the player can click or interact with) are capable of having three AI sessions attached to them. 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. 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.

A knowledge agent is basically a collection of rules that is used to create a session. 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.

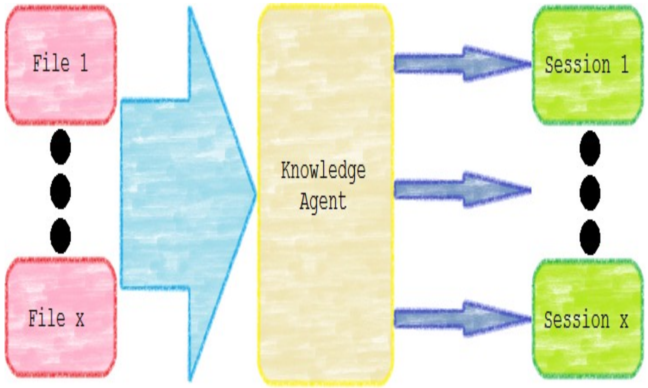


Figure 4: Knowledge agent and session creation process

4. AI DESIGN

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.

4.1 AI 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

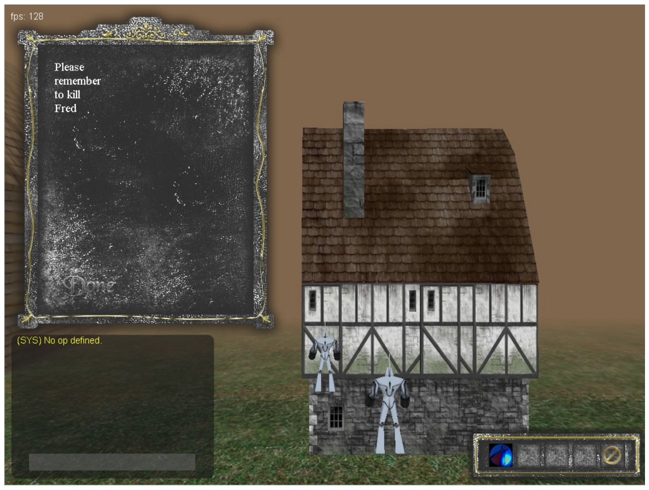
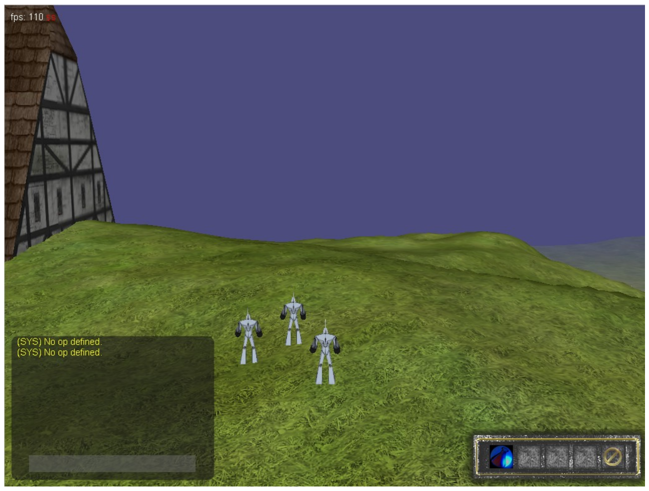


Figure 5: A dialog NPC's (left) response to a player(right) who's previously talked to him

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

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.



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

Figure 6: Two combat NPCs(left) following and attacking a player(right)

4.2 AI 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. Any entity is capable of using any combination of these sessions, including all or none of them.

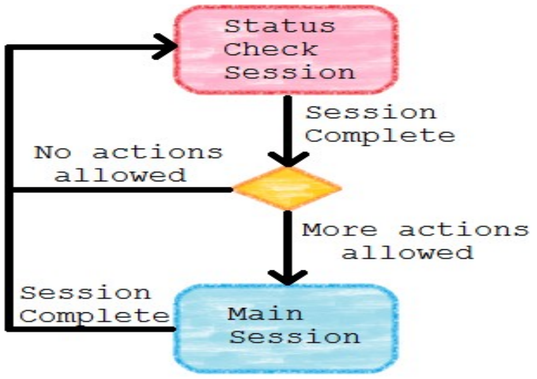


Figure 7: Status Session and Main Session flow

4.2.1 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.

```
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
```

Figure 8: Status Check Rule Example

4.2.2 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”.

```
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);
      if(detectionDist == null)
         throw new RuntimeException("Detection distance doesnt exist.");
      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
```

Figure 9: Detection Rule Example

4.2.2.1 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. Regardless of if the NPC has a target or not, when the "Detection" phase is over it moves onto the "Decision" phase.

4.2.2.2 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, or moving about the world according to some algorithm, until it finds something it can act on.

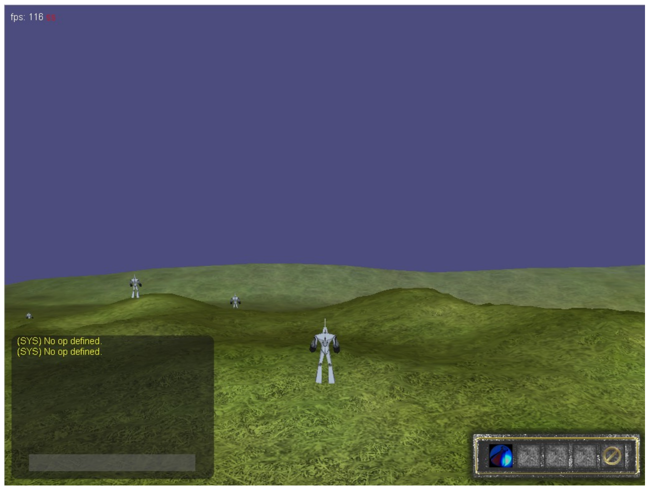


Figure 10: Two NPCs(back) randomly selecting points in an area and walking to them

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/spell/ability to use, whether it should attack or retreat, etc. Other unique behavior could also be implemented in the "Decision" phase as well.

```
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
```

Figure 11: Decision Rule Example

4.2.3 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.

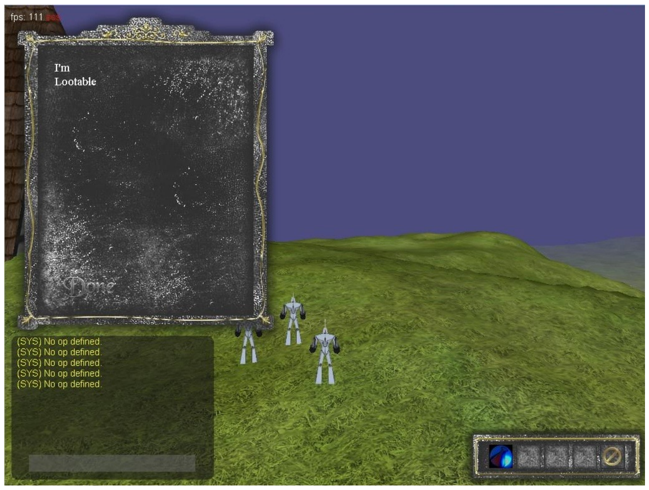


Figure 12: Right-clicking on a recently killed combat NPC opens this dialog instead of performing an attack

4.3 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.

4.3.1 Database Structure
In order to create rules, two tables are required in the database: the agent name table and the file assignment table. 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.

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

Figure 13: Agent Name 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('FireballAttacker.drl','basic_ai');
insert into drools_file (id, kbase_id) values('MeleeAttacker.drl','basic_ai');
insert into drools_file (id, kbase_id) values('Patrol.drl','elf32');
```

Figure 14: File Assignment Table Insert Examples

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. If a NPC does not use any AI for a session, the value "null" is passed in for that session.

```
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');
```

Figure 15: Assigning AI to NPC Examples

4.3.2 Rule Organization

Rule organization for spells/ability/item 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. There is generally only one spells/ability/item effect per file.

```
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
```

Figure 16: Item Rule Example

The dialog rules are equally simple, containing a list of rules that generate dialog based on various facts. 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.

```
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
```

Figure 17: Dialog Rule Example

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 Figure 9. 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.

4.4 Fact Management

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.

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 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');
```

Figure 18: Fact Table Insert Examples

5. CONCLUSION

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. However, 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 solution. 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. In the future, when all AI aspects have been well developed, we hope to address the issue of efficiency.

6. ACKNOWLEDGMENTS

The support of the National Science Foundation under Grant CNS-0938964 is gratefully acknowledged.

7. REFERENCES

[1] Epic Games, Inc. UDK – Unreal Development Kit. http://www.udk.com/

[2] Fletcher, G. H. and Lu, J. J. 2009. Education Human computing skills: rethinking the K-12 experience. *Commun. ACM* 52, 2 (Feb. 2009), 23-25. DOI= http://doi.acm.org.libproxy.lib.csusb.edu/10.1145/1461928.1461938

[3] Hsu, F., Campbell, M. S., and Hoane, A. J. 1995. Deep Blue system overview. In *Proceedings of the 9th international Conference on Supercomputing* (Barcelona, Spain, July 03 - 07, 1995). ICS '95. ACM, New York, NY, 240-244. DOI= http://doi.acm.org.libproxy.lib.csusb.edu/10.1145/224538.224567

[4] Jboss EAP. Drools: Business Logic integration Platform. http://www.jboss.org/drools/

[5] Laird, E. J. and Pottinger, D. Game AI: The State of the Industry, Part Two. Gamasutra – The Art and Business of Making Games. (Nov 2000). http://www.gamasutra.com/view/feature/3569/game_ai_the_state_of_the_.php?page=1

[6] Laird, J. E. 2001. It knows what you're going to do: adding anticipation to a Quakebot. In *Proceedings of the Fifth international Conference on Autonomous Agents* (Montreal, Quebec, Canada). AGENTS '01. ACM, New York, NY, 385-392. DOI= http://doi.acm.org.libproxy.lib.csusb.edu/10.1145/375735.376343

[7] Pilgrim, R. A. 1995. TIC-TAC-TOE: introducing expert systems to middle school students. In *Proceedings of the Twenty-Sixth SIGCSE Technical Symposium on Computer Science Education* (Nashville, Tennessee, United States, March 02 - 04, 1995). C. M. White, J. E. Miller, and J. Gersting, Eds. SIGCSE '95. ACM, New York, NY, 340-344. DOI= http://doi.acm.org.libproxy.lib.csusb.edu/10.1145/199688.199853

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

[9] Schulz, N. and Wiendl , V. Horde3D Next-Generation Graphics Engine. http://www.horde3d.org/home.html

[10] Streeting, S. et al. ORGE – Open Source 3D Graphics Engine. http://www.ogre3d.org/

[11] Wing, J. M. 2006. Computational thinking. *Commun. ACM* 49, 3 (Mar. 2006), 33-35. DOI= http://doi.acm.org.libproxy.lib.csusb.edu/10.1145/1118178.1118215