Rules of Engagement: Moving Beyond Combat-Based Quests

Anne Sullivan Michael Mateas Noah Wardrip-Fruin

Expressive Intelligence Studio
University of California, Santa Cruz
{anne, michaelm, nwf} @ soe.ucsc.edu

ABSTRACT
Computer role-playing games (CRPGs) are known for their strong narrative structure. Over time, quests have become one of the main mechanics for leading a player through the story. Quests are given to the player in the form of a set of tasks to complete with few, if any, options. The options given to the player instead often revolve around combat-oriented actions – requiring the player to engage in combat to progress through the storyline, despite player preference or game story that hints otherwise. We address this issue with the GrailGM, a run-time game master which offers quests and actions to the player based on their history and current world state.

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

General Terms
Design

Keywords
Role-playing games, quests, rule-based systems.

1. INTRODUCTION
Quests are one of the primary mechanics for leading players through stories in computer role-playing games (CRPGs) [39]. When looking at the genre, we found that while players have some interesting and meaningful choices during quest-driven gameplay, these choices are largely confined to combat. Conversely, it is typical for the game to be restricted to a pre-set narrative—the player moves through the experience, fulfilling checkpoints to advance the story. This imbalance is in large part due to the fact that there are highly formalized and flexible combat models found in the predecessors of CRPGs—table-top role-playing games and war games—but no such flexible formalization for story exists in current games.

INT3 2010, June 18, Monterey, CA, USA

Over time, combat systems have become quite robust, and many CRPGs have become heavily reliant on combat-based quests. In contrast, CRPG narratives mainly rely on three styles: linear, branching, and layered [27]. Linear stories have minimal, if any, choices for the player to make. Branching stories have some player choice, but often restricted to a few options with localized impact. Branching stories do, however, allow for multiple endings based on player actions. Layered stories have a main story structure that is either linear or branching, but include a number of side stories to add richness to the game world. However, these side stories are generally unrelated to the main storyline.

While CRPG combat is certainly enjoyed by some players, differences in player types [44,2] and player expectations shows that there is a desire for other quest options [40]. CRPGs present the player with rich, dynamic worlds and histories, but give the player little in the way of options in interacting with this world other than combat. In certain player types (e.g. Yee's Immersion player types or Bartle's Explorers) this can lead to frustration due to the lack of agency which we will discuss in more detail later in the paper.

To address this, we have begun implementing a system called the Grail Framework which consists of both a design-time and run-time components. The Grail Framework has goals at two levels. The first is to create a framework in which the authoring of high-level rules together with relatively-atomic pieces of traditional content can become a powerful way of authoring for CRPGs.

The second goal is to develop sets of rules that make the construction of new kinds of CRPG quests possible and tractable. By producing the relevant pieces of atomic content and their hooks into the rules, designers can create in particular: quests that take player history into account, have multiple acceptable action sequences for completion, and are driven by character relationships.

In particular for this paper, we will be focusing on the in-game component, the GrailGM. The GrailGM is a run-time game manager which uses the player's history and current world state to dynamically generate and alter quest structure. These changes changes allow player actions to have a discernable impact on the world (e.g. if the player makes an NPC angry with them, they may later get a quest to calm that person down.) The system also allows for more meaningful options given to the player by allowing multiple paths for completion of the quest which are dynamically generated based on player history.

In the rest of this paper, we will first describe the related work in the field. We discuss quests in depth, including the current state of

quests, as well as our proposed evolution. We then will discuss the Grail Framework, an overarching project which aims to address both the design-time and run-time issues of creating a CRPG with increased agency. Then we will turn our focus to the GrailGM, the run-time game manager, which uses the player's history and world state to perform dynamic quest composition.

2. RELATED WORK

There is a substantial amount of research related to interactive storytelling. In particular, we are focusing on research relating to narrative generation and quest generation in particular.

2.1 Narrative Generation

There are several paths taken within Narrative Generation. One path towards generated narrative is the research conducted on autonomous agents, such as the work of Pizzi et al. [30]. This work implements NPCs as autonomous agents, such that they react to the player in an interesting and believable manner. The narrative evolves based on the character's interaction with these agents, but there is no guarantee of coherence throughout the story.

Drama Management systems guide the user towards a more coherent story by adapting the options available to the player based on the player's actions. There have been a number of drama management systems including search-based drama management (SBDM) which was first proposed by Bates in 1992 [4] and developed by Weyhrauch in 1997 [43]. More recent work has generalized SBDM as declarative optimized drama management (DODM), for which search is one optimization method. These systems, such as the *Mimesis* architecture [45], the *Interactive Drama Architecture* [22], *Thespian* [34] and *U-Director* [26] all approach drama management in a unique way.

Both the beat-based DM in *Façade* [23] and the *PaSSAGE* system [38] employ a content-selection model of drama management. In *Façade*, the beat-based DM maintains a probabilistic agenda of dramatic beats. Each beat coordinates autonomous characters in carrying out a bit of dramatic action while supporting player interaction during the beat. Similarly, *PaSSAGE* contains a library of character encounters in a roleplaying game, dynamically selecting the next encounter as a function of a model of the player. Thue, et al. [37] expand this work into a delayed authoring system which aims to infer player state to offer a more player-specific experience.

The Grail Framework combines these two approaches by offering both semi-autonomous agents and a system of offering options to the player based on their previous actions within the game, but guided by authorial intent. The Grail Framework is most similar to *Façade* and *PaSSAGE* which both use content-selection model; however the Grail Framework is not necessarily constrained to one type of story "goodness." *Façade* uses Aristotelian dramatic rules to create a story that follows appropriate tension, while *PaSSAGE* and delayed authoring create user models to predict player preference. In the Grail Framework, the goals of the system are defined by the designer and can follow the rules used by either of these systems, or entirely different rules depending on the author.

Finally, Peinado and Gervás [29] began work on an interactive storytelling system that models a game master (GM) and player models based on the heuristics set forth by Robin Laws [20]. The system described was never completed to our knowledge, and as such, players were restricted to one of seven pre-made characters which had pre-created player models associated with them. In the Grail Framework, the heuristics used for choosing quests and quest solutions are not built into the system, but are instead specified by the designer.

2.2 Quest Generation

There are two major systems that deal with quest generation in particular. *Charbitat*[1], a game system consisting of generated terrain tiles with randomly placed components based on player actions, was expanded to include a quest generation system that worked in conjunction with the terrain generation. *Charbitat* implemented lock-and-key style quests based on spatial progression through the world. As the level was generated, a quest could be created on a new tile which used the world state as context for the goal of the quest. Unlike the Grail Framework, quests within *Charbitat* are generated without author input, but are based instead on the tiles the system is generating.

ScriptEase [24] is a designer tool created to work with the *NeverWinter Nights* [11] Aurora toolset [7]. ScriptEase follows a pattern-based approach to authoring, with many of the common designing tasks available as a pre-scripted selectable component in the tool. Many of the standard quests regularly found in CRPGs are available as a pattern in the quest library. These patterns are extensible so that a designer is not restricted to just the quests available in the library. Once created, these quests are playable within a *NeverWinter Nights* module, but unlike the Grail Framework, the quests are statically placed and do not change based on the player's actions.

3. QUESTS

Computer role-playing games (CRPGs) generally involve a storyline for the player to progress through. The stories are set up through introduction sequences or early game narration. Once the game begins, the player finds themselves thrust into the role of a character, playing through the game to experience the rest of the story.

While some games rely so heavily on their story that interaction is minimized (e.g. in *Xenosaga Episode 1*[25], there are 8 hours of cut scenes, including one with a built-in intermission), CRPGs more often integrate player actions within story progression. Not every action available will continue the story (e.g. talking to the beggar may lead to an interesting quip, but does not necessarily move the story along), so it is necessary to build in guidance for the player.

Over time, quests have begun to fill this role and, as mentioned above, are now one of the primary mechanics for leading a player through a CRPG story [39]. Quests are given to the player as a set of tasks to perform, often with one or more specified rewards for fulfilling the required tasks. Rewards are sometimes implicit or unstated, but tasks are generally explicitly declared. In many contemporary CRPGs such as *Dragon Age: Origins* [9] and *World of Warcraft* [14], the game has a built-in quest journal to track which quests are active (the player has accepted the quest and is working on it) and what set of tasks are required. In single-player CRPGs such as *Dragon Age: Origins* and *Oblivion* [6], the main reward for completing the quest is moving the story forward. In Massively Multiplayer Online Games (MMORPGs), there is rarely a central story—the driving story is that of character progression, with many more quests available—so the rewards are in the form of items, gold or faction gains (increased standing

with a specific group of people) as well as experience points. These are often explicitly stated in the quest journal.

3.1 Playable Quests

While quests allow for player interaction within the story, the interaction is often shallow with only localized consequences. By this, we mean that while the player may be given options to choose from, the options lead to the same general consequence, with just a few minor differences leading up to the outcome. The story often progresses along a fixed trajectory, with perhaps a chance of multiple endings for the player to experience. For instance, in *Dragon Age: Origins*, in the mage origin storyline, the player is given the option to betray a friend, or help the friend and lie to the leader of the mages. Whichever option is chosen, the end result is the same, with only minor dialog differences occurring within the game.

Another way to phrase this is that most current quests lack interesting or meaningful choices, and are therefore not "playable" [36]. To recap this definition, Rollings and Morris observe that an interesting choice is one in which "no single option is clearly [objectively] better than the other options, the options are not [subjectively] equally attractive, and the player must be able to make an informed choice" [31]. In addition, a meaningful choice is one in which the outcome of the choice is both "discernable and integrated" within the larger context of the game [32]. Not only should the player be able to have interesting choices, but their choices should have a noticeable (discernable) and significant (integrated) impact on the game world.

3.2 Reliance on Combat

Many CRPGs have a heavy reliance on combat [3]. This reliance likely stems from the robustness of the combat system, especially compared to the other possible player interactions.

In the majority of CRPGs, combat options allow for the most diversity of choice to the player. Even in most strongly story-based CRPGs such as BioWare's offerings (*Star Wars: Knights of the Old Republic* [12], *Baldur's Gate II* [8], *Dragon Age: Origins*, etc.) combat is a very well developed part of the game. Players are given multiple options within combat, not just along class specifics (spells for mages, ranged damage for archers) but also within their combat style. For instance, a mage in *World of Warcraft* may choose to use an ice spell to bind an enemy so that it cannot reach them, or they might choose to use a fire spell which causes the enemy to catch on fire and take damage over time. These options give the player interesting (damage via fire or movement impairment with ice) and meaningful (if I mess up I die) choices, making combat a very playable part of the game.

There are games that are exceptions to this, often considered the pinnacle of the CRPG genre [16,28] such as *Planescape: Torment* [13] and *Fallout 3* [5], which integrates non-combat solutions to a selection of its quests, allowing players to choose whether to take a traditional combat role or to fulfill one of the other supported solutions to complete the quest. But, given current tools, such multiple-solution quests are burdensome to implement and highly bug-prone. Wardrip-Fruin discusses these issues in regard to *Star Wars: Knights of the Old Republic*. When a player does not follow the expected path through the narrative, the fragile natures of the systems come to light. For instance, in playing, Wardrip-Fruin found a kidnapped son of a Dantooine family. After bringing that storyline to completion, he was later approached by the father when visiting the family compound and asked to find his son [41].

However, far more prevalent is the absence of player choice in quest actions, with a strong reliance on combat to provide the player with a feeling of options. Modern CRPGs do liberal borrowing of combat mechanics from other games, ranging from tabletop RPGs (e.g., *Dungeons & Dragons* [17]) to modern first-person shooters (as seen in *Mass Effect 2* [10]). The one common thread is the statistical interaction of characters (who progress in levels and abilities), their items, and a set of probabilities connected to external factors (e.g., enemies being faced, the configuration of space, etc). We don't deride this; we simply wish to make the story portion more satisfyingly playable.

3.3 Goal-based vs. Task-based Quests

As described above, combat reliance is compounded by the fact that quests generally have only one way to fulfill them. Jeff Howard describes a quest in literature as "a goal-oriented search for something of value" [18]. However, in CRPGs the player is presented a set of chores to complete to progress the story. The activities of the quests (mostly combat) are supported by a playable model, and result in progressions that depend on how you play (increasing stats in particular weapon use, increasing magic ability, etc.), but the narrative itself is fixed. This leads to quests that have merely a veneer of story—a story coating attempting to give meaning to a set of required tasks the player must complete.

In such quests, which we refer to as *task-based* quests, the list of tasks provides the player with the collection of non-optional actions that must each be completed in order to complete the quest. In contrast, a *goal-based* quest presents an end point, or goal, for the player to achieve. The player chooses, within the constraints of the game world and mechanics, how to reach the given goal.

An example of a task-based quest that could be found within many RPGs is the quest to save a farm from wolves. A farmer will give a player a task of killing the wolves to save the farm. The player must kill the wolves in order to receive the reward for completing the quest, regardless of class or role-playing preferences. In contrast, a goal-based quest would simply explain to the player that there are wolves killing the local livestock. The player would then be allowed options for completing the quest. They could still kill the wolves if they choose, but other options would be available, perhaps specific to certain character abilities or histories. For example, any character might have the option of creating better fencing, while a druid or ranger might help restore the local deer population so the wolves no longer have to hunt livestock. In this way, the player is able to make an interesting choice (no choice is clearly better than the other) based on in-game class, race, history, or personal preference. With the inclusion of these choices having a meaningful effect on the game (e.g. different quest rewards, game world evolution, or even affecting future quest and solution availability), quests become playable.

The existence of player choice in how quests are completed is the key difference between goal-based and task-based quests. Early adventure games such as *The Secret of Monkey Island* [21] and *King's Quest* [33] games often presented the puzzles within the game as a quest for the player to fulfill: "Become a pirate," "Save the mayor," "Save the land." While on the surface these seem like goal-based quests, there was in fact no player choice involved in quest completion. This could easily lead to frustration where the player would be searching for the solution the designers had

chosen for each quest, even though there were other options that made sense to the player but were not supported.

For example, in *Monkey Island* there exists the simple goal of getting past some deadly piranha poodles. At this point in the game, the player has become a sword master, but they are not allowed to fight the poodles, they must go find the exact item required to pass the dogs. The player has a chunk of meat, but this alone is not exactly what is required. Using the meat with grog (potent alcohol) does not work, but instead the player must eventually realize that they need to use the meat with a small flower they (hopefully) found while walking through the woods to drug the meat. Until the player stumbles upon this solution, they cannot progress further in the game.

This situation is at heart caused by the fact that these quests were presented as goal-based quests, but were in actuality task-based quests with opaque specifications. A true goal-based quest allows for interesting player choice, where there are multiple ways to fulfill the quest, and one solution is not obviously better than the others.

4. AGENCY

Moving to goal-based quests does not mean allowing the player to take every action imaginable. That would not only be intractable but does not actually increase agency, as described below. Instead we wish to use the concept of goal-based quests to increase the agency afforded to the player. Wardrip-Fruin, et al. describe agency as "a phenomenon involving both player and game, one that occurs when the actions players desire are among those they can take (and vice versa) as supported by an underlying computational model" [42]. By giving players choices without consequence to the quest and story progression, some designers are actually decreasing agency within their games—the "story coating" of the quests suggests possibilities for action that are not supported by the underlying rule system.

To increase agency requires matching the options the designers make available to the player with those the player expects based on the game narrative. This requires connecting the quest and story progression of the world to an underlying computational model, and structuring the experience such that the player develops an understanding of the computational model—enabling them to make choices with consequences based on the dramatic possibilities of the world.

Since many CRPGs take place in fantasy settings, it is useful to explicitly examine how agency fails in such settings. In a fantasy setting, players have knowledge and expectations of the world and characters derived from literary sources (e.g. Tolkien), movies, and previous fantasy games. For example, a player in *World of Warcraft* may have background knowledge about elves, such as that they are nature-loving and agile. Some of the dramatic possibilities stemming from this background knowledge are backed up by the game system—for example, elves are allowed to be archers (agile) or druids (nature-loving) and these choices in turn affect combat resolution options.

However, the current design and implementation strategies for quests do not support many of the dramatic possibilities arising from the player's background knowledge. Since quests are fixed lists of tasks, the quest progression is the same regardless of whether the player is playing a fighting character, a healing character, or a stealthy character. Further, quest progressions tend not to depend on special skills the player develops as their character levels up, even though these skills are given names that reference the background fantasy knowledge. For instance, a druid in *World of Warcraft* receives abilities called "Gift of the Earth Mother," "Barkskin," and "Wild Growth," but these skills, or any other druid-specific spells, are never required for quest completion.

Our research is motivated by this mismatch. We are interested in approaches that will make it possible to dynamically alter and generate portions of quest structures to support dramatic possibilities for a wide range of characters while avoiding the bug-prone brute-force methods currently in use.

5. THE GRAIL FRAMEWORK

The Grail Framework is designed to address the issues discussed above. The system encompasses both authoring tools and an in-game system called the GrailGM (Grail Game Manager). The Grail Framework in its entirety is designed to create a framework in which the designer is able to author high-level rules together with relatively-atomic pieces of traditional content. By not requiring the author to use traditional scripting methods to create quests, they are able to focus on the task of designing, as opposed to programming.

The Grail Framework also allows for the development of new kinds of CRPG quests. As the designer creates content and rules, the Grail Framework is able to dynamically combine these in new and interesting ways. The designer is able to maintain authorial control over the world via designer goals—similar to a Game Master presiding over a table-top role-playing game. Quests and solutions available to the player are shaped by the player's own history—details such as where the player has traveled, who they have talked to and the types of relationships with these NPCs, and how they have solved previous quests.

The role of the GrailGM is to be a stand-in Game Master for the designer while the game is running. In some ways, the GrailGM is similar to a Drama Management system such as DODM [35]; instead of manipulating plot points in the game story, the GrailGM uses the player's history and the designer's goals to select the quests available to the player. As the player learns more about the world, GrailGM updates what quest goals and actions are available. In this way, the world is reacting dynamically to the player's movement throughout the world, giving the player a higher sense of agency.

We are currently in the process of implementing the GrailGM portion of The Grail Framework. In the rest of the paper, we will discuss the current status of the GrailGM.

6. GRAILGM

The GrailGM is the run-time Game Master portion of the Grail Framework. Figure 1 provides a diagram of how the GrailGM interacts with the game.

As the player takes action inside the game world (e.g. talking to NPCs) the game updates the world and player states. The GrailGM contains recognizers, in the form of rules, which fire when the world/player is in a particular state. The particular states that are necessary are specified by the designer as part of the rules.

The Quest Manager uses rules to filter through the quest library, to find quests that are appropriate for the player given the state of the game. It also monitors the list of active quests to see if any quests have been completed. The Quest Manager uses this

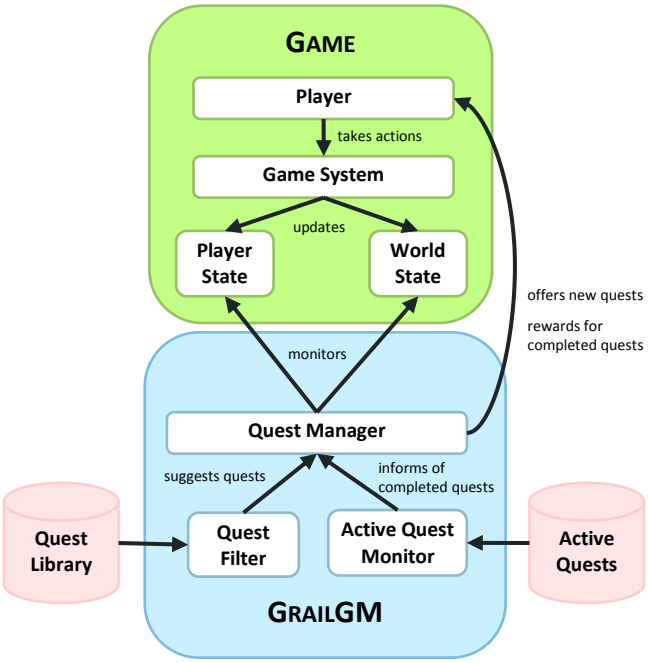

Figure 1: The interconnection between the game and the GrailGM.

information to inform the player of new quests, and to reward the player of quests that have been completed.

When a quest is chosen, it is removed from the quest library, and when an active quest is completed, it is removed from the list of active quests.

The GrailGM decomposes quests into different parts, storing each part as a separate entity within the quest library. The parts of a quest that are stored are:

- the quest goal: what the player needs to fulfill to complete the quest
- the available actions the player may take to complete the quest
- the reward for completing the quest
- non-player characters (NPCs) who are able to give the quest
- dialog options available to the NPCs involved in the quest

We use Drools [19], a forward-chaining inference based rules engine as the backbone of the GrailGM. Facts (relational data tuples) are asserted as working memory elements and these are matched against rule conditions. We have chosen to use a rule-based system as it naturally supports dividing quests into smaller component parts (NPCs, dialog, quest goals, quest solutions, etc.) and allows for authoring rules for dynamically recombining the pieces. This allows us to move away from scripting every possible recombination which would quickly become intractable and bug-prone.

Player state—consisting of knowledge known by the player, as well as player history—and world states are asserted as facts. Each part of a quest is represented as a rule, with designer-specified preconditions stored as the patterns the GrailGM uses to match to asserted facts. This will be described in more detail in the following sections. The GrailGM filters possible quests and actions through the pre-conditions based on player history, appropriate NPCs, and current world state.

Our game world is currently a text-based interface, with the GrailGM offering a list of actions to the player as a numbered list. Quests are automatically chosen and assigned to the player by the GrailGM; this will be changed in future versions.

We have chosen to use a text-based interface for our prototype system as it is the quickest route to showing a working system without the unnecessary complications a graphical interface would entail. Because of this, our actions are currently large granularity, for instance: "Talk to beggar," "Explore forest," and "Kill orc" would traditionally be comprised of multiple actions on the player's part; we have combined them into a singular action.

6.1 Example Scenario

For illustrative purposes, we will be using the following quest as an example for our system. Forest Song is a town situated in the Eyrewood Forest. The player arrives and, upon talking to the inhabitants, they learn that Forest Song was named for all the bird song that filled the forest. Unfortunately, the birds are no longer singing, and the townspeople are quite saddened by this. Following up, the player learns that the birds have quit singing because wild cats are scaring the birds. It is then up to the player to deal with the situation.

In a standard CRPG setting, the player would be required to talk to a specific NPC who would explain the problem: the birds are not singing. This NPC would then tell the player how to fix the issue, and assign the quest (or perhaps send the player to another NPC who would then assign the quest.) The quest would be in the form of "explore the Eyrewood Forest" or "talk to the town shaman." Once this task had been accomplished, the quest would be completed, and the player would now know about the wild cats. Likely, the player would then be dispatched to take care of the menace by killing a specific number of them to thin their numbers.

Within the GrailGM, the quest would be instead represented as a set of quest goals: "Figure out why Forest Song inhabitants are sad," "Find out why birds aren't singing," and "Save the birds." A list of possible actions associated with each quest is also represented. For instance, the actions associated with the "Find out why birds aren't singing" quests include: "Talk to an NPC who studies the birds," "Talk to an NPC who is a town elder," and "Explore the bird nesting grounds." The player chooses an action which would either lead to a new subquest or complete the active quest and start the next quest in the series.

We will now show how "Save the Birds" is specifically represented within the GrailGM.

6.2 Quest Goals

Within the GrailGM, quest goals are specified as a rule. A quest that is possible to be given to the player is defined as a ValidQuest. The pre-conditions are specified in the left-hand side in the "when" clause. Asserting the quest as valid is accomplished on the right-hand side in the "then" clause.

Save the Birds is represented below. In this example, we are stating that the following conditions must be true for the quest to be valid:

- The player must be in Forest Song
- The player must know about the cats scaring the birds

- they must not have completed the "Save the Birds" quest
- The "Save the Birds" quest cannot currently be active.

```
rule "Quest: Save the Birds"
when
 Location(npc == PLAYER,
            location == FOREST_SONG)
 Knowledge(npc == PLAYER,
            Knowledge == Intimidating(CATS, BIRDS))
 not CompletedQuest(
            questGoal == SAVE_BIRDS)
 not ActiveQuest(
            questGoal == SAVE_BIRDS)
then
    insert ( new
            ValidQuest(SAVE_BIRDS));
end
```

We track completed quests so the designer may create quest chains, and so quests are not repeated. We also use the relation type Intimidating in this example. While it would be possible to hard code CATS_SCARING_BIRDS, we plan to use complex relationships more in the future so we have represented it as a standalone fact.

If a quest "when" clause is matched, the "then" clause is executed. In this case, we assert that "Save the Birds" is a valid quest and it should be added to the list of possible options for the player.

Consequences of quest completion are specified in separate rules.

```
rule "Quest: Save the Birds complete"
when
 $activeQuest : ActiveQuest(
            questGoal == SAVE_BIRDS)
 not Intimidating(npc1 == CATS, npc2 == BIRDS)
then
 insert (new
            CompletedQuest(SAVE_BIRDS));
 retract ($activeQuest);
end
```

Here we first check that "Save the Birds" is the active quest. If it is, the GrailGM tests whether the cats are still intimidating the birds. If the birds are no longer scared by the cats, then "Save the Birds" is added to the completed quests, and retracted as the active quest. Quests are completed by the player performing actions, in this case, an action which retracts the fact Intimidating(CATS, BIRDS). We will look at actions next.

6.3 Actions

Once a quest goal has been found, the GrailGM then evaluates which actions are available for the player. All valid actions are presented to the player, and not all actions move the player towards completion of the active quest (e.g. "Talk to Beggar" may lead the player to a new subquest where they learn about the dark, seedy underbelly of Forest Song). Other actions are available only when specific quests are active.

Actions also take the form of rules. The rule conditions can take the form of player class restrictions, locations visited, or knowledge required by the NPC or player. If the conditions are true, then the action is asserted as a valid action, and presented to the player in a list for them to choose from.

Continuing our example, the player is on the "Save the Birds" quest. An example action rule for talking to the cats' friend would take the following form:

```
rule "Action: Talk to Cats' Friend"
when
 $npc : NPC(location == FOREST_SONG)
 Friend(npc1 == PLAYER, npc2 == $npc)
 Friend(npc1 == $npc, npc2 == CATS)
 Class(npc == PLAYER, class == DRUID)
then
 insert(new ValidAction(
            TalkTo(PLAYER,$npc,
            Intimidating(CATS,BIRDS))));
end
```

In this example, the preconditions for this action are a bit more complex. The player must be a druid class, and they must be a friend of a friend of the Forest Song cats. This friend of a friend could be the leader of the cats, another druid, or even the king of the forest. If these conditions are true, then a possible action is that the player can talk to the cats' friend about the scared birds. Presumably the cats' friend would be able to coax the cats to find new hunting grounds or to cease terrorizing the birds.

Again, consequences of completed actions are specified as separate rules.

```
rule "Carry out Action: Talk to Cats' Friend"
when
  $actionChosen : ActionChosen(action ==
            TalkTo(npc1 == PLAYER, $npc : npc2, subject ==
            Intimidating(CATS, BIRDS)))
 Friend(npc1 == $npc, npc2 == CATS)
then
 retract (Intimidating(CATS, BIRDS));
 retract ($actionChosen);
end
```

Here we see that if the player has chosen the action to talk to the friend of the cats, then we retract the fact that the cats are intimidating the birds. This will then trigger the rule we discussed above that handles the quest completion.

6.4 NPCs

Non-player characters or NPCs are used to add life to the world. These are game characters that the player can interact with, and are controlled by the game system. In our system, NPCs are used to convey information as well as start quests, but instead of a one to one mapping of NPC to quest as is found in most CRPGs, there is a many to one mapping where many NPCs may be available to introduce a quest or give information to the player.

Each NPC is represented as a set of facts in working memory. These facts include location, race, class or job, faction, and knowledge. These facts are used to match quests or actions to a specific NPC. For instance, if we were to create the cat leader mentioned above, they may look like this:

```
insert(new Location(CAT_LEADER, FOREST_SONG))
insert(new Friend(CAT_LEADER, CATS))
insert(new Class(CAT_LEADER, CAT))
insert(new Knowledge(CAT_LEADER, Intimidating(CATS, BIRDS)))
```

6.5 Dialog

Dialog is one of two major mechanics for information discovery by the player (exploration being the other). In practice, quest designers use tools to tie quest steps to pieces of dialog. This can require the designer to use programmatic logic to create dependency maps in the form of large if-else statements. In the GrailGM, dialog is not tied to a particular NPC. Instead it is treated as a rule similar to actions and quest goals. In this way it can be matched to multiple NPCs that fit the specified criteria, allowing the player to not be forced to talk to specific characters.

```
rule "Dialog: I was meditating in the forest and I saw some wild
cats chasing the birds!"
when
 $npc : NPC(name != PLAYER)
 Location(npc == $npc,
            location == FOREST_SONG)
 Class(npc == $npc, class == DRUID)
 Knowledge(npc == $npc,
            knowledge == Intimidating(CATS, BIRDS)
 not Knowledge(npc == PLAYER,
            knowledge == Intimidating(CATS, BIRDS)
then
    insert ( new
            ValidDialog($npc, MEDITATING_CATS_BIRDS));
end
```

Since dialog is tied to a TalkTo action, the consequences of the dialog are tied to the action itself.

7. FUTURE WORK AND CONCLUSION

We have presented the current progress of The Grail Framework, in particular the GrailGM. Currently the system allows for decomposing quests into a quest goal and actions, and decoupling this from NPCs and dialog. This allows for reassembling the quest in new and interesting ways based on the player history as well as the current state of the world.

In future iterations, we plan to change the way quests are assigned to the player, giving them more options as to which quests they accept. Additionally, more information will be stored when a quest becomes active. By storing the action or NPC that triggered the quest, we can use this information in later quests or dialog. For instance, if the player talks to the cat leader to complete the "save the birds" quest, it would be possible to later receive a quest from the cat leader as a request to return the favor.

This also leaves room for exciting extensions. In modern CRPGs, there is often a notion of faction; groups of NPCs that the player can earn favor with. However, more personal relationships are much more difficult to model and are often heavily scripted and confined to just a few major players. In the GrailGM we have begun adding the notion of relationships; NPCs and players can be friends, or they can be intimidated. This is just the beginning of a more complex set of relationships that can be represented within the GrailGM. For example, it is possible to encode as a rule "The enemy of my enemy is my friend."

These relationships can then be used for matching quest goals or actions. Instead of the designer being required to create a series of quests where the player becomes friends with an NPC, then finally given a quest where they are asked to betray them, instead a general betray quest goal could be created. The GrailGM would then be able to match it to any NPC that is friends with the player, no longer requiring a forced friendship.

Currently, the system uses a text-based interaction which allows us to create some shortcuts in regards to player actions. However, we plan to move to a graphical interface in the future. At this point, instead of a hard-coded selection of actions, we will instead need to recognize actions, using more complex recognizers similar to those used in a drama management system.

Once the system is further along, we plan to evaluate our system using the similarities between our system and drama management, we plan to leverage the testing methods introduced by Chen, et al. [15] for testing authorial leverage. The goal of the testing will be to show that a given amount of rule authoring within the system results in an amount of quest variation that, when expressed as hard-coded if-else rules, would not be tractable to specify.

In conclusion, this system will offer a new way of addressing quests within computer role-playing games. It enables authors to tractably craft options that connect to player understandings of dramatic possibilities and the game system, allowing the player's past actions to affect the quests and actions available in a wide variety of ways. In this way, quests become playable; the player is presented interesting and meaningful choices not just within combat, but within the story itself.

8. REFERENCES

1 Ashmore, Calvin and Nitsche, Michael. The Quest in a Generated World. In *Situated Play: International Conference of the Digital Games Research Association* (Tokyo, Japan 2007).

2 Bartle, Richard. Hearts, clubs, diamonds, spades: Players who suit MUDs. *Journal of Online Environments*, 1, 1.

3 Barton, Matt. *Dungeons and Desktops: The History of Computer Role-Playing Games*. A K Peters, Ltd., Wellesley, 2008.

4 Bates, Joseph. Virtual Reality, Art, and Entertainment. *Presence: The Journal of Teleoperators and Virtual Environments*, 1, 1 (1992), 133-138.

5 BETHESDA GAME STUDIOS. *Fallout 3*. Bethesda Softworks. 2008.

6 BETHESDA SOFTWORKS. *Elder Scrolls IV: Oblivion*. 2006.

7 BIOWARE. *Aurora Toolset*. 2002. ships with NeverWinter Nights.

8 BIOWARE. *Baldur's Gate II: Shadows of Amn*. 2000.

9 BIOWARE. *Dragon Age: Origins*. Electronic Arts. 2009.

10 BIOWARE. *Mass Effect 2*. Electronic Arts. 2010.

11 BIOWARE. *NeverWinter Nights*. 2002.

12 BIOWARE. *Star Wars: Knights of the Old Republic*. 2003.

13 BLACK ISLE STUDIOS. *Planescape: Torment*. 1999.

14 BLIZZARD ENTERTAINMENT. *World of Warcraft*. 2004.

15 Chen, Sherol, Nelson, Mark J., Sullivan, Anne, and Mateas, Michael. Evaluating the Authorial Leverage of Drama Management. In *Proceedings of AAAI Spring Symposium, Interactive Narrative Technologies 2* (Stanford 2009).

16 GAMEPRO. *The 26 Best RPGs*.

http://www.gamepro.com/article/features/207777/the-26-best-rpgs-page-1-of-4/. 2008. Last accessed March 2010.

17 Gygax, Gary and Arneson, Dave. *Dungeons & Dragons (d20 System)*. TSR, Wizards of the Coast. 1974-current.

18 Howard, Jeff. *Quests: Design, Theory, and History in Games and Narratives*. A K Peters, Ltd, Wellesley, 2008.

19 JBoss. *JBoss Rules*. 2009.

20 Laws, Robin D. *Robin's Laws of Good Game Mastering*. Steve Jackson Games, 2002.

21 LUCASFILM GAMES. *The Secret of Monkey Island*. 1990.

22 Magerko, Brian. Story Representation and Interactive Drama. In *Proceedings for Artificial Intelligence for Interactive Digital Entertainment* (Marina del Rey 2005).

23 Mateas, Michael and Stern, Andrew. Façade: An Experiment in Building a Fully-Realized Interactive Drama. In *Game Developers Conference, Game Design Track* (San Jose, CA 2003).

24 McNaughton, Matthew, Cutimisu, Maria, Szafron, Duane, Schaeffer, Jonathan, Redford, James, and Parker, Dominique. ScriptEase: Generative Design Patterns for Computer Role-Playing Games. In *International Conference on Automated Software Engineering* (Linz, Austria 2004).

25 MONOLITH SOFT. *Xenosaga Episode 1*. Namco Bandai. 2003.

26 Mott, Bradford W. and Lester, James C. U-director: A Decision-Theoretic Narrative Planning Architecture for Storytelling Environments. In *International Joint Conference on Autonomous Agents and Multiagent Systems* (Hakodate, Japan 2006).

27 Paul, R., McNeill, M., Charles, D., McSherry, D., and Morrow, P. Real-Time Planning for Interactive Storytelling. In *Proceedings of the 9th Irish Workshop on Computer Graphics* (Dublin 2009), 89-94.

28 PCGAMER. *PCGamer Top Ten*.

http://www.pcgamertop100.com/editorial/10-1. Last Accessed March 2010.

29 Peinado, Federico and Gervás, Pablo. Transferring Game Mastering Laws to Interactive Digital Storytelling. In *International Conference on Technologies for Interactive Digital Storytelling and Entertainment* (Darmstadt, Germany 2004).

30 Pizzi, David, Cavazza, Marc, and Lugrin, Jean-Luc. Extending character-based storytelling with awareness and feelings. In *Proceedings for International Conference on Autonomous Agens and Multiagent Systems* (Honolulu 2007).

31 Rollings, Andrew and Morris, Dave. *Game Architecture and Design*. New Riders Publishing, Berkeley, CA, 2003.

32 Salen, Katie and Zimmerman, Eric. Game Design and Meaningful Play. In Raessens, J. and Goldstein, J., eds., *Handbook of Computer Game Studies*. MIT Press, Cambridge, MA, 2005.

33 SIERRA ENTERTAINMENT. *King's Quest*. 1984-1998.

34 Si, Mei, Marsella, Stacy C., and Pynadath, David V. Thespian: using multi-agent fitting to craft interactive drama. In *Proceedings for the International Joint Conference on Autonomous Agents and Multi-Agent Systems* (Utrecht 2005).

35 Sullivan, Anne, Chen, Sherol, and Mateas, Michael. From Abstraction to Reality: Integrating Drama Management into a Playable Game Experience. In *Proceedings of AAAI Spring Symposium, Interactive Narrative Technologies 2* (Stanford 2009).

36 Sullivan, Anne, Wardrip-Fruin, Noah, and Mateas, Michael. QuestBrowser: Making Quests Playable with Computer-Assisted. In *Proceedings for Digital Arts and Culture* (Irvine 2009).

37 Thue, David, Bulitko, Vadim, and Spetch, Marcia. Making Stories Player-Specific: Delayed Authoring in Interactive Storytelling. In *Joint International Conference on Interactive Digital Storytelling* (Erfurt, Germany 2008).

38 Thue, David, Bulitko, Vadim, Spetch, Marcia, and Wasylishen, Eric. Interactive Storytelling: A Player Modelling Approach. In *Artificial Intelligence and Interactive Digital Entertainment* (Palo Alto, CA 2007).

39 Tosca, Susana. The Quest Problem in Computer Games. In *Proceedings of Technologies for Interactive Digital* (Darmstadt 2003).

40 Tychsen, Anders, Tosca, Susana, and Brolund, Thea. Personalizing the Player Experience in MMORPGs. In *Proceedings of the Third Technologies for Interactive Digital Storytelling and Entertainment* (Darmstadt 2006), 277-288.

41 Wardrip-Fruin, Noah. *Expressive Processing: Digital Fictions, Computer Games, and Software Studies*. The MIT Press. 2009.

42 Wardrip-Fruin, Noah, Mateas, Michael, Dow, Stephen, and Sali, Serdar. Agency Reconsidered. In *In Breaking New Ground: Innovation in Games, Play, Practice and Theory: International Conference of Digital Games Research Association* (London 2009).

43 Weyhrauch, Peter W. *Guiding interactive drama*. Carnegie Mellon University, Pittsburgh, 1997.

44 Yee, Nick. Motivations for Play in Online Games. *CyberPsychology & Behavior*, 9, 6 (January 2007), 772-775.

45 Young, R. Michael and Reidl, Mark. Towards an Architecture for Intelligent Control of Narrative in Interactive Virtual Worlds. In *Proceedings for the International Conference on Intilligent User Interfaces* (Miami 2003), 310-312.