Proceedings of the Second International Workshop on Procedural Content Generation in Games, pp. 1-8, Bordeaux, France, 2011.

A Prototype Quest Generator Based on a Structural Analysis of Quests from Four MMORPGs

Jonathon Doran
Dept. of Computer Science & Engineering
University of North Texas
jhd@unt.edu

Ian Parberry
Dept. of Computer Science & Engineering
University of North Texas
ian@unt.edu

ABSTRACT

An analysis of over 750 quests from four popular RPGs (Eve Online, World of Warcraft, Everquest, and Vanguard: Saga of Heroes) reveals that RPG quests appear to share a common structure. We propose a classification of RPG quests based on this structure, and describe a prototype quest generator based on that classification. Our aim is to procedurally generate quests that are complex, multi-leveled, and plausible to players of RPGs. We analyze a nontrivial quest from Everquest and one from our prototype quest generator for comparison.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem solving, control methods, and search—*Plan execution, formation, and generation*; I.6.8 [**Simulation and Modelling**]: Type of Simulation—*Gaming*

General Terms

Algorithms, Experimentation

Keywords

Quest, role-playing game, RPG, MMORPG, NPC, procedural content generation, planning, intractability, Java, Prolog, BNF.

1. INTRODUCTION

A *quest* is a player task commonly found within role playing games where the player is challenged to complete goals in return for some reward. From an author's point of view, the quest provides many of the challenging elements of game play, and provide players with concrete goals [14]. From a player's point of view, the quest is a narrative element that informs them about the world and allows the player to gain knowledge and power. Dramatic events may be portrayed through the dialog and actions of characters involved with a quest. While authors seek exposition and players seek entertainment, these goals are not necessarily mutually exclusive.

Procedural quest generation clearly has the potential to increase the variability and replayability of games. This in turn can lead to an increase in player interest in these games, as there never comes a point where the player has seen everything or done everything in the game. The seemingly limitless quests that might be generated could cause one to dismiss quest generation as an intractable problem. However, an analysis of over 750 human-authored quests for Eve Online, World of Warcraft, Everquest, and Vanguard: Saga of Heroes shows these quests to have significantly less structural variety than one might expect. Without any artificial restrictions on quest structure, human authors nonetheless create works with shared structure, changing details such as setting, but preserving the relationship between actions. We have been able to exploit this common structure and demonstrate a prototype system that procedurally generates random quests appropriate for use in RPGs.

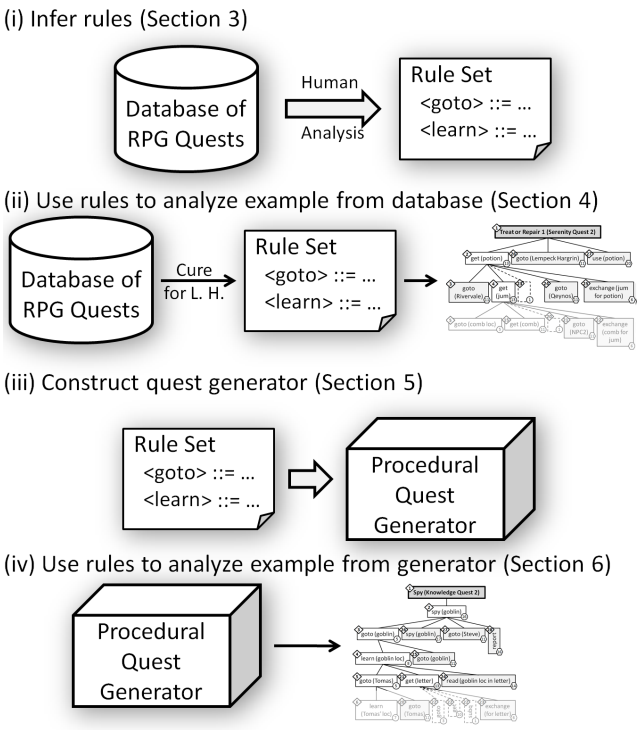

Figure 1: Roadmap for the rest of this paper.

PCGames '11, June 28, 2011, Bordeaux, France

The main part of this paper is divided into five short sections. In Section 2 we consider related work. In Section 3 we describe the results our structural analysis of quests from four MMORPGs (see Figure 1(i)), a preliminary version of which appeared in [11]. In Section 4 we give an example of the structural analysis of a quest from Everquest (see Figure 1(ii)). In Section 5 we describe a prototype quest generator (available online at [10]) based on the structural analysis (see Figure 1(iii)). In Section 6 we analyze a quest from our quest generator (see Figure 1(iv)).

2. RELATED WORK

Previous research on RPG quest analysis falls into two categories: structural and functional. Autonomous generation of quests involves not only a suitable form for a quest, but a means for the generator to know when to generate a quest and the ability to ensure that the quest makes sense in the current game state.

Some classification studies have been performed [6, 12, 13], but these do not necessarily have the goal of autonomous generation of quests. Sullivan has also classified player actions, and has came to similar conclusions [13]. Ashmore and Nitsche [7] propose procedural quest generation using key-lock structure quests, rather than NPC interactions. This structure has potential for autonomous generation, but lacks the sense of purpose found in quests derived from NPC goals. Jill Walker [15] classifies World of Warcraft quests into exploration and combat quests, which is significant as it is an early attempt to classify quests. We believe our classification is more general, in that it can express additional types of quests. Dickey [9] provided a classification system which appears suitable for autonomous generation, but does not claim to provide complete coverage for RPG quests nor does it address how quests could be generated based on this system. Our decomposition is closely related to world state, which makes it fairly straightforward to select a quest type which modifies some piece of state. Aarsath discussed quests in general, although using a somewhat liberal definition which could be applied to first-person shooters [6].

Quest generation is a special case of planning, as one may consider a quest to consist of an initial state, a goal, and a set of actions players must perform to reach the goal. Planning in general is NP-complete, as a planning problem may be restated as a boolean satisfiability problem (whether an action is present at a particular stage of a viable plan) [8]. As a result, exhaustive search may be required to determine even if a plan exists.

The search space for non-trivial domains also grows very quickly, as planning is PSPACE-hard. Therefore while general planning algorithms are the most capable technique for generating quest plans, they are impractical. We avoid this limitation by abandoning the general planning problem in favor of one with restrictions. We are able to do this without loss of generality as our analysis of human generated quests showed these quests to have structural patterns which occurred in predictable situations. Each of these patterns has an implicit precondition and postcondition, and as long as these conditions are not violated a generator is free to elaborate or introduce new subquests.

3. STRUCTURAL ANALYSIS

For our structural analysis of RPG quests (refer back to Figure 1(i)), we examined over 750 quests from Everquest, World of Warcraft, Vanguard: Saga of Heroes, and Eve Online to determine whether common structures were present. Table 1 gives the distribution of quests from the four games. Quest descriptions were obtained from Allakhazam [1],the MMODB Quest Database [2], Silky Venom [3], Thottbot [4], and the Eve Info Mission Database [5]. Each quest description consists of NPC dialog, an explanation of the actions players are required to perform, and the reward for completing the quest.

Motivation	Quests	Percent
Everquest	266	35%
Vanguard	137	18%
World of Warcraft	205	27%
Eve Online	145	19%

Table 1: Distribution of quest origins.

Motivation	Description
Knowledge	Information known to a character
Comfort	Physical comfort
Reputation	How others perceive a character
Serenity	Peace of mind
Protection	Security against threats
Conquest	Desire to prevail over enemies
Wealth	Economic power
Ability	Character skills
Equipment	Usable assets

Table 2: NPC motivations.

We observed the following patterns. Quests fall into a set of 9 distinct underlying motivations best described by the following list of nouns: Knowledge, Comfort, Reputation, Serenity, Protection, Conquest, Wealth, Ability, and Equipment (see Table 2). Our use of motivations is novel, and we believe essential for ensuring that quests appear intentional and appropriate rather than randomly generated. An NPC's motivation is a statement of the most important concern the NPC has, and the quest is intended to address this concern. We assume that as game events unfold a particular NPC's motivations will change, especially if players successfully complete quests for this NPC. The distribution of motivations observed was not uniform (see Table 3); we assume that the designer could in principle change the distribution of motivations over time to affect the flavor of the

Motivation	Quests	Percent
Knowledge	138	18.3%
Comfort	12	1.6%
Reputation	49	6.5%
Serenity	103	13.7%
Protection	137	18.2%
Conquest	152	20.2%
Wealth	15	2.0%
Ability	8	1.1%
Equipment	139	18.5%

Table 3: Distribution of observed NPC motivations.

game by preferring one form of quest motivation over another.

The NPC delivering the quest to the player does so using from 2-7 specific strategies specific to its motivation. Each strategy can be described by a verb-noun pair, for example, "kill pests", "steal supplies", or "rescue NPC". The full list of strategies with their corresponding motivations from Table 2 is given in Table 4.

For each strategy there is a sequence of from 1 to 6 actions that the player must undertake in sequence to implement the strategy. Each action can be either a simple atomic action that be achieved by the player (see Table 5, or can be expressed recursively as a sequence of other actions or action variants (see able 6). We will express quest structure in the form of a grammar in which the terminal symbols are atomic actions.

Actions can therefore be expressed as an infinite set of trees, the leaves of which are atomic actions that can be performed by the player, and the internal nodes of which represent tasks that must be achieved along the way. A list of the leaves in the order visited by pre-order traversal of the tree gives the sequence of atomic actions required to achieve the quest. For example, Table 6 gives a partial action structure in the form of a grammar in Backus-Naur Form. An RPG quest then starts with a choice of NPC motivation:

<QUEST> ::= <Knowledge> | <Comfort> |
<Reputation> | <Serenity> |
<Protection> | <Conquest> |
<Wealth> | <Ability> | <Equipment>

A Comfort quest, for example, (lines 7 and 8 of Table 4) can then be described as:

<Comfort> ::= <Obtain Luxuries>
<Obtain Luxuries> ::= <get> <goto> <give>

Next, in Section 4, we will use these rules to analyze the quest "Cure for Lempeck Hargrin" from Everquest as an example of how our rules can describe quests from existing RPGs. After a brief description of our quest generator in Section 5, we will in Section 6 give an example of a quest generated from the rules.

4. QUEST ANALYSIS EXAMPLE

To demonstrate how a sample human-designed quest from a popular RPG fits into our classification scheme (refer back to Figure 1(ii)), consider the quest "Cure for Lempeck Hargrin" from the game Everquest. The quest begins in Qeynos with NPC0, who asks the player to travel to a remote town and obtain an ingredient needed to cure an ailing NPC named Lempeck Hargrin. The player is required to travel from Qeynos to Rivervale, talk to NPC1 there, follow his directions to visit NPC2 who is injured, and give NPC2 bandages in return for honeycombs. The honeycombs are then taken to NPC1 in Rivervale and traded for honey jum. This honey jum is then taken to the originating NPC0 in Qeynos and used to make a curative potion. The player takes this potion to Lempeck Hargrin who exchanges a scythe for the potion and is cured. The player delivers the donated scythe to NPC0 and is then granted a reward (the Shining Star of Life).

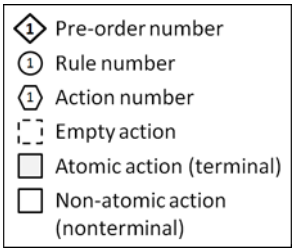

Figure 2: Key to Figures 3 and 5.

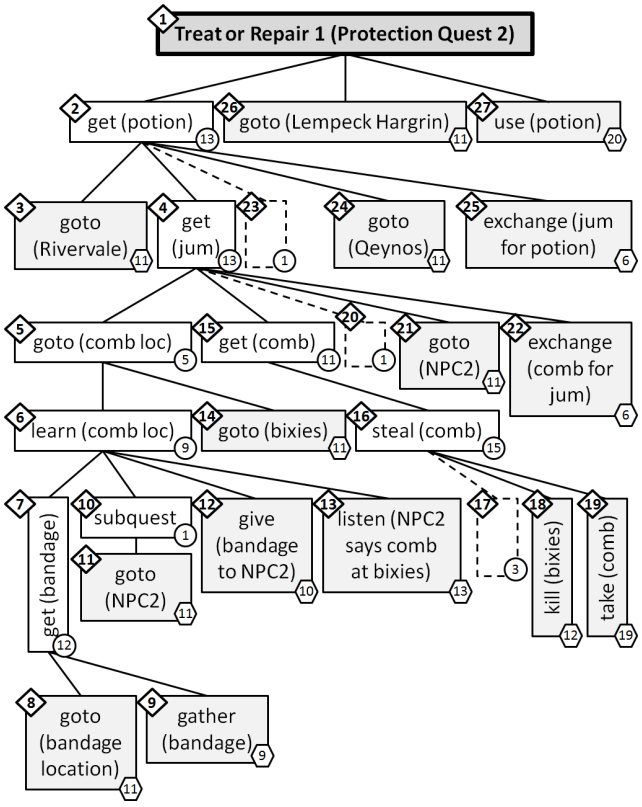

Figure 3: Analysis of "Cure for Lempeck Hargrin" from Everquest.

Since the primary motivation for asking the player's help is the need to cure an NPC, this quest is motivated by Protection. The strategy chosen by the NPC assigning this quest to the player is Treat or Repair (1). This strategy breaks down into the sequence of actions (see Table 4):

<Serenity> ::= <Treat or Repair (1)>
<Treat or Repair (1)> ::= <get> <goto> use

The tree is then recursively filled out to Figure 3. Figure 2 gives the key to Figure 3. Nonterminal symbols are shown in white boxes with circled numbers showing the corresponding production rule from Table 6. Terminal symbols are shown in gray boxes with numbers in hexagons showing the corresponding action from Table 5. Numbers in diamonds show the order in which actions are performed, obtained from a pre-order traversal of the tree. These numbers correspond to the numbers in the following analysis:

1. <Treat or Repair (1)>: Receive quest from NPC1. Consult Table 4, Line 18 to get the sequence of actions.

Motivation	Strategy	Sequence of Actions
Knowledge	Deliver item for study	<get> <goto> give
	Spy	<spy>
	Interview NPC	<goto> listen <goto> report
	Use an item in the field	<get> <goto> use <goto> <give>
Comfort	Obtain luxuries	<get> <goto> <give>
	Kill pests	<goto> damage <goto> report
Reputation	Obtain rare items	<get> <goto> <give>
	Kill enemies	<goto> <kill> <goto> report
	Visit a dangerous place	<goto> <goto> report
Serenity	Revenge, Justice	<goto> damage
	Capture Criminal(1)	<get> <goto> use <goto> <give>
	Capture Criminal(2)	<get> <goto> use capture <goto> <give>
	Check on NPC(1)	<goto> listen <goto> report
	Check on NPC(2)	<goto> take <goto> give
	Recover lost/stolen item	<get> <goto> <give>
	Rescue captured NPC	<goto> damage escort <goto> report
Protection	Attack threatening entities	<goto> damage <goto> report
	Treat or repair (1)	<get> <goto> use
	Treat or repair (2)	<goto> repair
	Create Diversion	<get> <goto> use
	Create Diversion	<goto> damage
	Assemble fortification	<goto> repair
	Guard Entity	<goto> defend
Conquest	Attack enemy	<goto> damage
	Steal stuff	<goto> <steal> <goto> give
Wealth	Gather raw materials	<goto> <get>
	Steal valuables for resale	<goto> <steal>
	Make valuables for resale	repair
Ability	Assemble tool for new skill	repair use
	Obtain training materials	<get> use
	Use existing tools	use
	Practice combat	damage
	Practice skill	use
	Research a skill(1)	<get> use
	Research a skill(2)	<get> experiment
Equipment	Assemble	repair
	Deliver supplies	<get> <goto> <give>
	Steal supplies	<steal>
	Trade for supplies	<goto> exchange

Table 4: Strategies for each of the NPC motivations from Table 2 using actions from Table 5.

	Action	Pre-condition	Post-condition
1.	ε	None.	None.
2.	capture	Somebody is there.	They are your prisoner.
3.	damage	Somebody or something is there.	It is more damaged.
4.	defend	Somebody or something is there	Attempts to damage it have failed.
5.	escort	Somebody is there	They will now accompany you.
6.	exchange	Somebody is there, they and you have something.	You have theirs and they have yours.
7.	experiment	Something is there.	Perhaps you have learned what it is for.
8.	explore	None.	Wander around at random.
9.	gather	Something is there.	You have it.
10.	give	Somebody is there, you have something.	They have it, and you don't.
11.	goto	You know where to go and how to get there.	You are there.
12.	kill	Somebody is there.	They're dead.
13.	listen	Somebody is there.	You have some of their information.
14.	read	Something is there.	You have information from it.
15.	repair	Something is there.	It is less damaged.
16.	report	Somebody is there.	They have information that you have.
17.	spy	Somebody or something is there.	You have information about it.
18.	stealth	Somebody is there.	Sneak up on them.
19.	take	Somebody is there, they have something.	You have it and they don't.
20.	use	There is something there.	It has affected characters or environment.

Table 5: Atomic actions.

	Rule	Explanation
1.	<subquest> ::= <goto>	Subquest could be just to go someplace.
2.	<subquest> ::= <goto> <QUEST> goto	Go perform a quest and return.
3.	<goto> ::= ε	You are already there.
4.	<goto> ::= explore	Just wander around and look.
5.	<goto> ::= <learn> goto	Find out where to go and go there.
6.	<learn> ::= ε	You already know it.
7.	<learn> ::= <goto> <subquest> listen	Go someplace, perform subquest, get info from NPC.
8.	<learn> ::= <goto> <get> read	Go someplace, get something, and read what is written on it.
9.	<learn> ::= <get> <subquest> give listen	Get something, perform subquest, give to NPC in return for info.
10.	<get> ::= ε	You already have it.
11.	<get> ::= <steal>	Steal it from somebody.
12.	<get> ::= <goto> gather	Go someplace and pick something up that's lying around there.
13.	<get> ::= <goto> <get> <goto> <subquest> exchange	Go someplace, get something, do a subquest for somebody, return and exchange.
14.	<steal> ::= <goto> stealth take	Go someplace, sneak up on somebody, and take something.
15.	<steal> ::= <goto> <kill> take	Go someplace, kill somebody and take something.
16.	<spy> ::= <goto> spy <goto> report	Go someplace, spy on somebody, return and report.
17.	<capture> ::= <get> <goto> capture	Get something, go someplace and use it to capture somebody.
18.	<kill> ::= <goto> kill	Go someplace and kill somebody.

Table 6: Action rules in BNF.

2. <get>(potion): You need to get the potion. You've been told to travel to Rivervale and ask for honeyjum. Apply Rule 13 from Table 6.
3. goto(Rivervale): You know where it is, so go there. This is Action 11 from Table 5.
4. <get>(honeyjum): You ask for honeyjum, but you are told to get honeycomb. Apply Rule 13 from Table 6.
5. <goto>(honeycomb location): You want to go to the honeycomb, but you don't know where that is. Apply Rule 5 from Table 6.
6. <learn>(honeycomb location): First you need to learn where the honeycomb is. You know that NPC2 knows. Apply Rule 9 from Table 6.
7. get(bandage): NPC2 is injured and needs a bandage. Go to a bandage store and get it. Apply Rule 12 from Table 6.
8. goto(bandage location): You know where to get a bandage, so go there. This is Action 11 from Table 5.
9. gather(bandage): Gather a bandage, may need to purchase it or just pick it up. This is Action 9 from Table 5.
10. Your subquest is to go to NPC2. Apply Rule 1 from Table 6.
11. You go to NPC2. This is Action 11 from Table 5.
12. give(bandage to NPC2): You give the bandage to NPC2. This is Action 10 from Table 5.
13. listen(NPC2): NPC2 says to get honeycomb from bixies. This is Action 13 from Table 5.
14. goto(bixies): You go to where the bixies are. This is Action 11 from Table 5.
15. <get>(honeycomb): You want to get the honeycomb from them. Apply Rule 11 from Table 6.
16. <steal>(honeycomb): You must steal it from them. Apply Rule 15 from Table 6.
17. (Empty <goto>, you're already there.)
18. kill(bixies): You kill the bixies. This is Action 12 from Table 5.
19. take(honeycomb): You take their honeycomb. This is Action 19 from Table 5.
20. (No subquest.)
21. goto(NPC2): You take the honeycomb to NPC2. This is Action 11 from Table 5.
22. exchange: NPC2 exchanges honeycomb for honeyjum. This is Action 6 from Table 5.
23. (No subquest.)
24. goto(Queynos): You return to NPC1 in Qeynos. This is Action 11 from Table 5.
25. exchange: NPC1 exchanges honeyjum for potion. This is Action 6 from Table 5.
26. goto(Lempeck Hargrin): Deliver the potion. This is Action 11 from Table 5.
27. use(potion): Quest over. He's cured. This is Action 20 from Table 5.

5. QUEST GENERATOR

We have used our quest structure to create a generator capable of creating random quests (refer back to Figure 1(iii)). It starts with an NPC motivation from the list in Table 2 and a random number generator seed. We accept seeds from the user to allow quests to be reproduced for documentation. The seed 0 is used to request that the generator be seeded using system state (such as time) resulting in an unpredictable quest.

From the motivation, the generator consults a list of root strategies such as the list of alternatives for the "Ability" motivation shown in Table 4. The generator selects one of these strategies and creates a quest that addresses the motivation provided by the user. The sequence of actions can be viewed as leaves in a tree, with the root representing the entire quest. Most of these actions may be replaced by subquests made up of the same set of actions. By repeatedly replacing nodes with subquests, a tree of arbitrary depth and complexity may be created. Individual quests differ from each other in the order of actions the player is expected to perform (the leaves of the tree). Random trees will tend to have unique leaves, resulting in unique quests. The choice between alternate rules in Table 6 is made based on what the player is assumed to know and what the state of the game is at that time, and if alternatives remain, then one of them is chosen at random. Thus for example, the generator avoids having the player go on a quest for something they already have, or assumes knowledge that cannot possibly be obtained by the player given the current game state.

Specific details of the quest are randomly assigned as the quest tree is being expanded, resulting in differences between quests with the same structure. A simple delivery quest may be repeatedly used when one changes the item being delivered, the source of the item, and the item's destination. Games such as Everquest and World of Warcraft routinely reuse this same structure with different details, creating a large number of similar but distinct quests. Our generator is capable of recreating this variety of quests using a single structure. The popularity of this replacement technique in actual games demonstrates how much may be accomplished by just changing details, and suggests that a generator that changes both details and structure will produce useful content.

Our generator is written in Prolog, and we have provided a Java applet front-end as shown in Figure 4. Our choice of the Prolog language was based on Prolog's ability to automatically backtrack and try alternative solutions to a problem. This means that if our generator is unable to successfully expand part of a tree, that it will automatically abandon the effort and try something different.

Our generator's output consists of a terse tree of actions demonstrating quest structure, and a narrative describing the players actions while completing the quest. The narrative includes simple boilerplate dialog for NPCs, which provides a better feel for how the quest will appear than a pure description of actions.

6. GENERATED QUEST EXAMPLE

Now we will analyze a sample quest from our procedural quest generator (refer back to Figure 1(iv)). Our example starts with an NPC named Steve who is motivated by the need for Knowledge from Table 2. Steve picks the strategy Spy from Table 4, which requires a sequence of actions starting at <spy> from Table 6. The quest generator randomly chooses to fill out the quest as shown in Figure 5, using the production rules from Table 6 (refer back to Figure 2 for the key to Figure 5). As before, numbers in diamonds show the order in which actions are performed, obtained from a pre-order traversal of the tree. These numbers correspond to the numbers in the following analysis:

1. <Spy>: Receive quest from Steve. Consult Table 4,

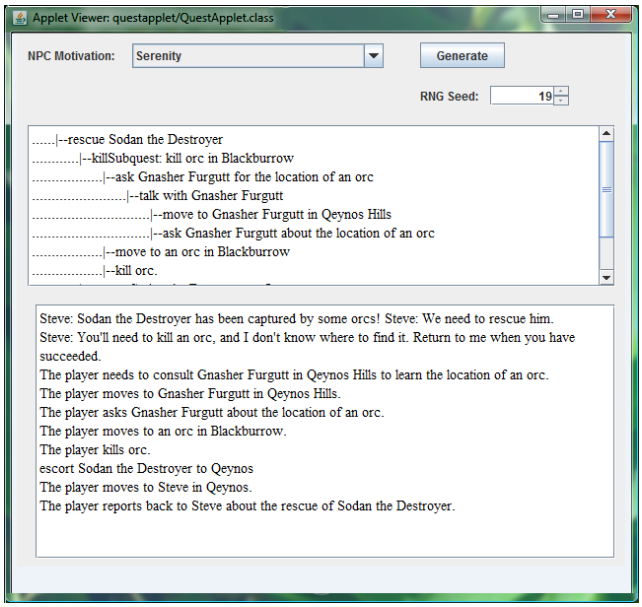

Figure 4: Screen shot of a prototype quest generator

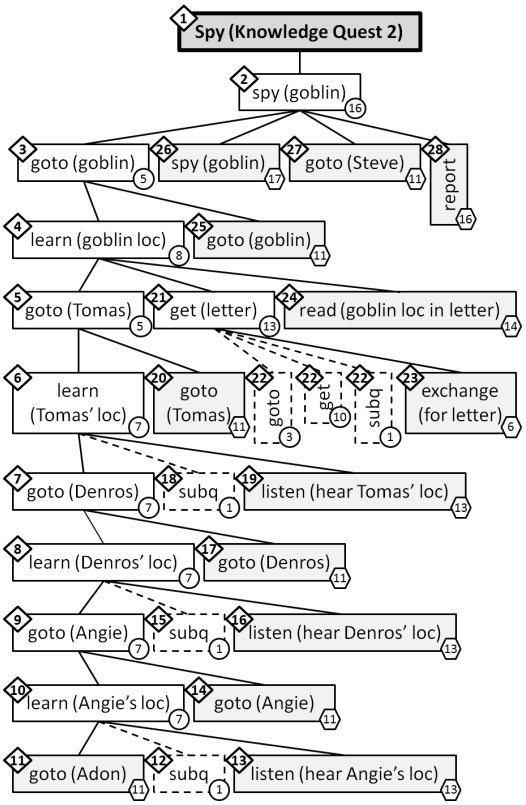

Figure 5: Analysis of a procedurally generated spy quest.

Line 2 to get the sequence of actions.

2. <spy>(goblin): Steve has told you to spy on a goblin. Apply Rule 16 from Table 6.
3. <goto>(goblin): You need to go to the goblin. But you don't know where the goblin is. Apply Rule 5 from Table 6.
4. <learn>(goblin location): You need to find out where the Goblin is. You know that Tomas has a letter that will tell you that. Apply Rule 8 from Table 6.
5. <goto>(Tomas): You must go to Tomas, but you don't know where he is. Apply Rule 5 from Table 6.
6. <learn>(Tomas' location): You need to find out where Tomas is. You know that Denros knows that. Apply Rule 7 from Table 6.
7. <goto>(Denros): You need to go to Denros, but you don't know where he is. Apply Rule 5 from Table 6.
8. <learn>(Denros' location): You need to find out where Denros is. You know that Angie knows that. Apply Rule 7 from Table 6.
9. <goto>(Angie): You need to go to Angie, but you don't know where she is. Apply Rule 5 from Table 6.
10. <learn>(Angie's location): You need to find out where Angie is. You know that Adon knows that. Apply Rule 7 from Table 6.
11. goto(Adon): You go to Adon. This is Action 7 from Table 5.
12. (Adon does not give you a subquest.)
13. listen: Adon tells you Angie's location. This is Action 9 from Table 5.
14. goto(Angie): You've finished learning Angie's location, so you go there. This is Action 7 from Table 5.
15. (Angie does not give you a subquest.)
16. listen: Angie tells you Denros' location. This is Action 9 from Table 5.
17. goto(Denros): You've finished learning Denros' location, so you go there. This is Action 7 from Table 5.
18. (Denros does not give you a subquest.)
19. listen: Denros tells you Tomas' location. This is Action 9 from Table 5.
20. goto(Tomas): You've finished learning Tomas' location, so you go there. This is Action 7 from Table 5.
21. <get>(letter): You want to get the letter from Tomas. Tomas tells you he will sell it to you. Apply Rule 13 from Table 6.
22. (You don't need to go anyplace, get anything, or perform a subquest to get it.)
23. exchange: You buy the letter from Tomas. This is Action 3 from Table 5.
24. read: You read the letter and it tells you where the goblin is. This is Action 10 from Table 5.
25. goto(goblin location): You go to where the goblin is. This is Action 7 from Table 5.
26. spy(goblin): You spy on the goblin. This is Action 12 from Table 5.
27. goto(Steve): You go back to Steve, who gave you the quest. This is Action 7 from Table 5.
28. report: You tell Steve where the goblin is. This is Action 11 from Table 5. The quest is complete.

7. CONCLUSION AND FURTHER WORK

Based on a study of over 750 quests from four MMORPGs we have found that quests have a well defined structure. We have inferred motivations for NPC's to grant these quests,

and believe that using this information to control generation will lead to a greater sense of realism. We have a prototype quest generator based on our analysis at [10]. However, further work is needed before we can demonstrate its ability to generate quests equal in quality to hand-crafted quests.

8. REFERENCES

[1] Allakhazam MMO Quest Databases, Retrieved June 2009. `http://zam.com`.

[2] MMODB Quest Databases, Retrieved June 2009. `http://www.mmodb.com`.

[3] Silky Venom Vanguard Wiki, Retrieved June 2009. `http://wiki.silkyvenom.com/index.php/Main_Page`.

[4] Thottbot World of Warcraft Database, Retrieved June 2009. `http://thottbot.com`.

[5] Eve Info Mission Database, Retrieved October 2010. `http://eveinfo.com/missions`.

[6] E. Aarseth. From Hunt the Wumpus to EverQuest: Introduction to Quest Theory. In F. Kishino, Y. Kato, and H. Nagata, editors, *Proc. ICEC 2005, Lecture Notes in Computer Science*, volume 3711, pages 496–506, 2005.

[7] C. Ahsmore and M. Nitsche. The Quest in a Generated World. In *Proc. DiGRA 2007*, 2007.

[8] T. Bylander. Complexity Results for Planning. In *Proc. Twelfth International Joint Conference on Artificial Intelligence*, volume 1, pages 274–279. Morgan Kaufmann, 1991.

[9] M. Dickey. Game Design and Learning: A Conjectural Analysis of How Massively Multiple Online Role-Playing Games (MMORPGs) Foster Intrinsic Motivation. *Educational Technology Research and Development*, 55(3):253–273, 2007.

[10] J. Doran and I. Parberry. Quest Generation. `http://www.eng.unt.edu/ian/research/quests/`.

[11] J. Doran and I. Parberry. Towards Procedural Quest Generation: A Structural Analysis of RPG Quests. Technical Report LARC-2010-02, Laboratory for Recreational Computing, Dept. of Computer Science & Engineering, University of North Texas, May 2010.

[12] A. Sullivan. Gender-Inclusive Quest Design in Massively Multiplayer Online Role-Playing Games. In *Proc. 4th International Conference on Foundations of Digital Games*, pages 354–356. ACM, 2009.

[13] A. Sullivan, M. Mateas, and N. Wardrip-Fruin. QuestBrowser: Making Quests Playable with Computer-Assisted Design. In *Digital Arts and Culture 2009*, UC Irvine: Digital Arts and Culture 2009, 2009.

[14] A. Tychsen, S. Tosca, and T. Brolund. Personalizing the Player Experience in MMORPGs. *Lecture Notes in Computer Science*, 4326:253–264, 2006.

[15] J. Walker. *A Network of Quests in World of Warcraft*. MIT Press, 2007.