Multi-agent planning for interactive emergent narrative systems

by

Vincent Breault

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Master of Cognitive Science

in

M.Cognitive Science

Carleton University
Ottawa, Ontario

Abstract

Generation of narrative is a domain that is mostly exclusive to human authors. Most current narrative generators use planning to adapt or follow a human authored story or guideline. This work proposes an engine for Creation Of Novel Adventure Narrative (CONAN). It uses a procedural approach to story generation that uses series of quests to make a rich player – NPC (non-player character) interaction from which a structured narrative can emerge. The engine is tested on its ability to create quests, which are sets of actions that must be performed in order to achieve a certain goal, usually for a reward. Compared against human structural quest analysis, the current engine was found to be able to replicate the structure found in commercial video game quests.

Acknowledgements

All the gratitude in the world for the great help of Sebastien Ouellet for his help with programming and implementation, and my supervisor Dr. Jim Davies, as well as the Science of Imagination Laboratory members, without whose comments and input this would not have been possible. And a special thank you to my wife for supporting me all this time.

Table of Contents

List of Tables

List of Appendices

Introduction

The creation of media content has always been the domain of humans, be it for movies, music or video games. With advancement in computer technology and research, the creation of such content has seen a slight shift from the human authored to automatic computer generation. Using algorithms to procedurally create media can effectively alleviate some of the burden from artists when creating a new piece. According to Hendriks and Meijer (2013), Procedural Content Generation for Games (PCG-G) is the use of computers algorithms to generate game content, determine if it is interesting, and select the best ones on behalf of the players.

This type of generation becomes quite useful when trying to produce content for an industry that is more and more demanding in terms of content (Hendriks & Meijer, 2013). For instance, in the current market, game development costs are extremely high as the demand for highly complex games requires the work of many artists and many hours to be met. For instance, the Massively Multiplayer Online Role Playing Game (MMORPG) World of Warcraft has a total of 30,000 items, 5,300 creatures with which to interact and 7600 quests and has an estimated budget of twenty to one hundred and fifty million dollars for a single game (Hendriks & Meijer, 2013). An engine capable of offloading this task by automatically generating such content would be invaluable to the industry, as it would greatly reduce development costs by reducing the amount of work that has to be done by humans.

Procedural generation (PG) in video games is currently used in a variety of different subfields to generate content. Indeed, where artists have previously been used to create everything, some of the work has been offloaded to automation on various level of design such as graphics (e.g., textures in Lefebvre & Neyret, 2003) or vegetation through such programs as SpeedTree (Re, Abad, Camahort, & Juan, 2009). The creation of environment and maps has also

seen its share of procedural generation algorithms for the generation of buildings (Martin, Lim, Colton & Browne, 2010; Kelly & McCabe, 2007), road networks (Sun, Yu, Baciu & Green, 2002) and maps (Hartswood, Zook, Das & Riedl, 2011; Togelius, Preuss & Yannakaki, 2010). PG can also be observed in the automatic or custom creation of more abstract elements such as ships or weaponry (Hastings, Guha & Standley, 2009), Non-Player Character (NPC) behaviour (Orkins, 2004) or other such system behaviour.

Another area where PG is used to generate abstract elements is narrative generation. According to Doran and Parberry (2010), characters in games give "quests" to the player character (PC). A quest is a set of actions that must be performed in order to achieve a certain goal, usually for a reward. They are usually provided by non-player characters within the game and are embedded in a piece of narrative that makes the sequence of actions make sense given the NPC and the current world state. They are used to engage the player in interacting with the NPC and for her to observe and learn about the NPC and the world in which the story takes place. Some quests are simple, but some are complex. For example, the quest "Cure for Lempeck Hargrin" that Doran (2010) uses for comparison has 27 steps, while others might have only a few. Given the structural similarities between the outputs of AI planning agents and quests, planning has seen a lot of use in the PG of stories. People have used planning and machine learning to modify previous human authored stories (Boyang & Riedl, 2010), to control NPC behaviour and the overarching story (Teutenberg & Porteous, 2013) or to generate fixed, step-by-step quests such as the ones found in an MMORPG (Doran & Parberry, 2010).

In order to build systems that reason about narrative, we must first understand how narratives are defined. In standard systems, the story is fully scripted by a human author and presented to the player as it unfolds. This limits the capabilities for adaptation to player

preferences and has very low replay value. In order to counteract this limitation, systems and frameworks have been designed to generate stories either dynamically, as it unfolds (Brenner, 2010; Teutenberg & Porteous, 2013) or at the start of a session, by selecting story elements, ordering them and presenting them to the audience (Doran & Parberry, 2011; Riedl, Thue & Bulitko, 2008; Hong & Riedl, 2012). According to Riedl and Young (2010), the two main aspects of narrative are the logical causal progression of a plot, meaning that the events that occur in the narrative obey the rules of the world in which it takes place, and character believability, defined as the perception by the player that the characters act in a coherent fashion, without negatively influencing the suspension of disbelief. This means that the events in the story must appear logical and the agents must appear intentional (Dennett, 1989) in order for a piece of narrative to make sense.

Planning is a good computational equivalent of stories, given the ordering and causality of actions in a plan sequence (Riedl & Young, 2010). Prince (1987) has given a definition of narrative as "the recounting of a sequence of events that have a continuant subject and constitute a whole". Therefore for a sequence of events to be considered a story, it must follow the general direction in which it started, and keep adding events that are coherent with past events.

The causality of events in a story, the relationship between the temporally ordered events that change the world state, is a property of narratives that ensures its coherence and a continuing subject (Chatman, 1993). Therefore, for a sequence of events to be considered a story, it must maintain coherent causal relationships between the events. At the same time, character believability is dependent on the coherent causal relationships between the character's attributes known to the player, such as the character's personality or desires, and said characters' actions.

This means that when deciding upon sequences of actions, characters need to make plans according to their own goals in order to appear intentional (Riedl & Young, 2010).

As mentioned previously, plans and narratives have many things in common. Given Prince's (1987) description of a narrative, one can see that while narratives are sequences of coherent and cohesive events that describe a series of changes in the world over the course of the story, plans consist of temporally ordered operations that transform the world state with each step, making them closely related. A planning system, given a story world and actions pertaining to said story world in the form of events, will create an artefact not unlike a story. Furthermore, partially ordered plans allow freedom in the order in which events occur if they are not causally related.

Planners require a domain theory containing all the possible events that can happen, an initial state which, in the case of narrative planning, would be the story world and all that it contains, and a goal situation, which again in the case of narrative planning, is what the final state of the world has to be for the goal to be achieved. The task of the planner is thus to find a sequence of actions that links the initial state to the goal state. With this, we are provided with an initial situation for a story, the events that unfold and the finale of the piece of narrative. In this sense, one can see that creation of narrative can be considered a problem solving activity, in that the solution of the problem is the plan itself, and can thus be performed by computers algorithms.

Although advances were made in this field, procedural generation of game narrative still has flaws. One of them is the fact that for classical planning algorithms, though they will successfully find a sequence of actions leading from the initial state to the goal state, their sequence is in no way guaranteed to make sense with the characters in the story. According to Riedl and Young (2010), the planning problem itself does not concern itself with character

believability. This means that if the goal state has a princess locked up, there is nothing preventing the planning system from having the princess lock herself up and considering this a valid plan, given that it satisfies the requirements of the author. Furthermore, many systems, such as Riedl, Thue and Bulitko's (2008), use human authored stories or human authorial intent to be able to create a coherent and cohesive story. These systems, called by Riedl and Young (2010) deliberative narrative systems, often use centralized reasoning in order to produce a narrative that satisfies the constraints and parameters intended by the human authors, also called authorial intent. In contrast to those systems are simulation based approaches, with a multi-agent simulation and distributed planning by and for each agent that simulate a story world. The system determines agents' actions depending on the current context and world state, solving the problem of making intentional agents (Teutenberg & Porteous, 2013). By simulating a world with intentional characters, believable interactions can emerge from the simulations (Aylett, 2006). Many systems using emergence also use director agents (Magerko, Laird, Assanie & Stokes, 2004) in order to guide the story, satisfy author goals and ensure interesting and well-structured performance of the simulation.

In his 2010 paper, Brenner states that plots often depend on plans failing or being thwarted and then being readjusted. Often times, in stories, multiple sub-stories or short events occur, the sum of which amounts to the overarching story. These sub-stories, in the context of games, can also be called quests. Many definitions of the term have been offered in the literature. Aarseth (2005) criticises Tosca (2003) as not being general enough and thus formulates the following definition 'a game with a concrete and attainable goal, which supersedes performance or the accumulation of points (…)' which can be reduced to 'a game which depends on mere movement from position A to position B'. Most games would have quests that include a number

of additional features not included in the reduced version such as characters and dialogue but they would still involve the necessity of moving from A to B. One might see how, through a series of A-to-B movements with a specific objective, a simple story (or quest) might take shape. For example, the quest 'go kill the dragon' can be translated as 'Move from A to B', 'Kill Dragon', 'Move from B to A'. Therefore, plans that involve movement through the space in order to fulfill an objective constitute valid quests, which, as has been noted above, are the building blocks of an overarching story.

In terms of content, quests can take many forms and shapes. Aarseth (2005) divides quests in three basic categories. The first is "place oriented," where the player has to move their avatar through the world to reach a target location with puzzles along the way, such as in Cyan Inc.'s *Myst*. Slightly less common are the "time-oriented" quests, where the task of the quest might simply be to survive for an interval of time. Last is the "objective oriented" quest where the task is to achieve a certain objective, such as bringing an item somewhere or taking it by force from an agent in the world. These basic categories can be combined, nested and serialized to produce more or less complex quests.

Given the fact that in simulation-based systems characters can appear intentional, and that interaction amongst them can appear believable, the emergence of a coherent story can exist if the audience is a human player interacting with the simulation instead of a director agent. I suggest that a coherent story will emerge from such an engine given a simulation-based system with believable characters that provide the human player with believable interactions in the form of quests. Furthermore, I suggest that the human player, through their interactions in the game, will be acting as a director agent. In director agent systems (Magerko, Laird, Assanie & Stokes, 2004; Laird & van Lent, 2001), the AI entity chooses interesting plot points and adjusts the story

to maximize its entertainment value. I suggest here that the player's choice of which quest to undertake within a large set of available quests, which sub-story to pursue, and which actions to perform in the world will ensure that the story elements, the quests as they unfold, will be interesting to the audience, the human player. This is so as the player represents both her own player model and a director agent, as they interact with the agents that they find the most interesting. Furthermore, Aarseth (2005) states that the best story-like device is the quest, which provides purpose to the naked space and exploration of the world. This means that the interactions with the agents, in the form of quests the player chooses according to their preferences, would produce interesting emergent narrative. That is, given a world of believable NPCs that act and seem to reason in a rational manner, the story emerges from the believable interactions of the PC with said NPC through quests, as the world progresses and events occur. The story itself would be the sequence of events and interaction performed by the player.

I thus suggest here an engine that relies very lightly on authorial intent in favour of a more open system where story direction is decided at runtime by the player and the agents. Instead of trying to make the engine itself write the entire story, I suggest letting the story emerge from player to non-player character interaction. The series of performed quests will create the overarching story as the player does more and more quests with the same agents, following their own player preferences in lieu of a player model enforced by an algorithm. The choice of which of the many available quests are undertaken, and thus what happens in the game world, by the player is what ensures the emergence of the story. The system will use realistic, intent-driven NPCs that generate plans to reach their goals according to the current world state. The intent based planning from these NPC is to create believable interactions as noted by Aylett, Dias and Paiva (2006) and again by Teutenberg and Porteous (2013). These plans are then

passed onto the player character as simple quests with an objective to fulfill. The resolution is open ended, meaning that although a specific plan has been created and hints given to the PC as to how to complete the given quest, no exact steps need be taken for the success of the quest so long as the objective is completed. I argue that these interactions with the NPCs in a persistent world and the persistence of the PC actions by themselves weave a narrative, without the direct writing of story by a human author. Such an engine would require very little in the ways of authoring. Indeed, once the setting of the story is created, the locations and the characters decided, the course of the story itself is unveiled as the player resolves the quests proposed to her by the engine.

The current work aims to create the proposed engine for the Creation Of Novel Adventure Narrative (CONAN) implemented in software. I will here implement and test the CONAN system on the created quests from the NPCs and the alteration of the world state as the interaction progresses. According to the theory explained above, the NPCs need to be able to provide pertinent quests to the player, the sequence of which, in a persistent world, must be coherent. Assuming that human authored quests are indeed relevant, the system must therefore be able to create similarly built quests. This experiment aims to create the quest generation engine CONAN in such a way that the quests it creates and offers to the player are similar to the ones a human author would. Furthermore, the CONAN engine is to create quests that are not only similar to the ones written by human authors but that are also relevant according to the current context of the game and the character giving out the quest in the world. I hypothesize that the CONAN system will be capable of creating quests similar to human authored quests and that these quests will be coherent, as defined by their relation to the current world state and the NPCs personalities.

Literature Review

Procedural generation has been used to create a variety of elements pertaining to all aspects of media. Game bits, such as textures, are images used in games for adding detail to geometry and models, and for giving a visual representation to game elements such as menus. These, amongst other things, can be generated procedurally through a variety of techniques such as rule based pattern creation and genetic algorithms (Lefebvre & Neyret, 2003; Whitehead, J., 2010; Sims, K., 1991). Other aspects, such as music (Bartle, 2003), vegetation (Re et al., 2009; Sims, K., 1991) and buildings (Kelly & McCabe, 2007; Greuter, Parker, Stewart & Leach, 2003) have also successfully been created in a procedural fashion, without the explicit use of human creators.

When it comes to story, narrative or otherwise giving context to gameplay, several different approaches have surfaced in the literature. Puzzles and quests are problems to which the player can find a solution based on previous knowledge or by systematically exploring the space of possible solutions embedded in the problem. One example of procedural generation of such content uses spatial progression through game space for generation of "key and lock puzzle" (Ashmore & Nitsche, 2007). They use spatial metaphors to tackle the problem of giving meaningful structure to randomly created space, as implemented in a prototype game they called "Charbitat". This world has a human authored overarching story and uses a procedurally generated spatial obstacle – solution to structure gameplay. In the game, the world is generated as the player explores it, following a tile based system with fauna, flora, and landmarks for each tile. The system uses these elements as obstacles, also called "locks". For instance a river might be one such lock. For each such obstacle, the system also generates a solution, to which the authors refer as "key". Following the above example, the key to crossing the river might be a

boat to be found somewhere on the map. The puzzle generator uses a set of rules as to which lock/key pair might be used and when and where to make them appear. For instance, when the player encounters a new area, the program goes through every lock/key combination to find the appropriate one according the rule set, such as there must be three locks of a certain type preceding the appearance of the key. These rules are user defined as to allow for different types of gameplay. The puzzle is thus finding out what the lock is and what/where the key might be. It is considered solved when the player moves to the next area. While this solution gives meaning to the generated space of a game, the narrative itself is still heavily reliant on human authored content.

While the above described engine generates puzzles, others have created quest-generating engines. Doran and Parberry (2011) used a structural analysis of human authored quests from popular Massively Multiplayer Online Role Playing Games (MMORPG) to create a quest generating engine using the extracted rules and commonalities. They defined quests as "a series of actions that benefit an NPC or faction in the game." Quests are motivated by the needs and desires of these NPCs or factions and players are typically rewarded for the completion of said quest. Through their analysis, they extracted rules on these quests. For instance, they noticed that all quests can be classified into initial NPC motivation, and each of these contain general motivation-specific strategies to attain goals which in turn are composed of sequences of actions for each strategy. Sequences of action are further defined as either atomic actions to be performed by the player or recursive sets of atomic actions, which have pre and post conditions. They found that NPC motivation can be classified in 9 categories; knowledge, serenity, wealth, comfort, protection, ability, reputation, conquest and equipment. For instance, a quest could have for motivation 'Protection', 'Attack threatening entities' as strategy and the action sequence

would be the following: <goto> damage <goto> report. In order to damage, "Something or someone is there" is a pre-condition and "It is more damaged" is a post condition. Using these rules, the engine uses a motivation and a strategy and will then generate sequences of actions by recursively adding to the tree of action sequence, allowing for arbitrary branching and depth of complexity.

The story of a game is often key in creating a good gaming experience. In their 2008 paper, Riedl, Thue and Bulitko approach the problem of automatic story generation through plotline adaptation, the difference being the use and modification of human authored content as opposed to starting from scratch. Their system takes a story as a plan-like representation of narrative for its causality and temporality of actions. The partial-order plans consist of events and their temporal and causal relations. In their system, quests are high level events composed of tasks and reward events, themselves composed of primitive events. This creates a hierarchy of events, from the very abstract and high level quest to the more concrete events that must be performed that is well suited for graph representation (e.g.: abstract witch-hunt quest contains a witch-hunt task which contains the action "throw water at witch"). The system also uses model of the player's preferences from previously selected quests. The algorithm thus takes the partially ordered and hierarchical plan and the set of quests to be included and rewrites the plot to meet the player model requirements. The engine then looks for flaws and dead ends in the ordered plan and modifies it using the quests from the set. This process allows for a single human-authored story to be used several times, being constantly modified through each iteration. It chooses its modification following the player's preferences taken from a player model. For example, if the player model knows that the player does not like hunting witches, it will remove this part of the story and replace it with something else that is knows the player finds interesting.

While this system may create many different stories from a single one, taking into account the player's preferences, it first requires a human-written story as well a human written database of story elements to be swapped in and out from the original story.

In their 2010 paper, Riedl and Young described their planning algorithm they call IPOCL, which stands for Intent-based Partial Order Causal Link. In order to create coherent overarching stories while retaining believable characters, they use character intentionality in the planning. The engine is provided with all elements in the world, including explicit representations of character intentionality, initial states and goals that must be reached, which define the story the authors intends to be told. From the end result, the engine performs an iterative backward search through the space to find actions that lead to the end. Their given example had in the final state a corrupt president. The action found that lead to this statement being true was the villain bribing the president. For each selected actions, the system looks for open conditions (such as the villain needing to have money to bribe or the villain having the intention of corrupting the president) and adds or removes actions in the sequence as necessary. They found that this ensures coherence in overarching story and in character actions as opposed to 'off the shelf' planning algorithm which are designed for a singular agent.

This algorithm has been used again by Haslum (2012) where he looked to prove Riedl's problem could be solved by standard planners. Through compilations such as pruning redundant actions, he has found the capabilities of standard planners to be enough to solve the problem and reduce processing time from 12.3 hours to 45 seconds.

A second approach to this problem is the use of multiple autonomous agents. In his paper, Brenner (2010) integrates planning algorithms into a framework suitable for narrative generation. They use a multi-agent planning system in which agents have explicit models of desire and

intentions in order to create appropriate plan schemes. They base themselves on the idea that plots depends on plans failing, being modified or otherwise foiled and then changed to adapt to a given situation ultimately leading to success or failure. The agents create plans that are dynamic, in the sense that planning and acting are interwoven during runtime. For this to be feasible in a multi-agent system, agents, also referred to as "character", are imbued with communication and perception models. They developed what they called a Distributed Continual Planning (DCP) method (DesJardins et al., 1999; Brenner & Nebel, 2009), in which agent's beliefs and goals are explicitly modeled. To do this, they used a multi-agent variant of the standard syntax for writing planning problems, PDDL (Planning Domain Definition Language) (Russel, Norvig, Canny, Malik, and Edward, 1995) called Multiagent Planning Language MAPL (Brenner & Nebel, 2009). MAPL has the advantage of its actions having a controlling agent who executes them. The plans are also partially ordered, using causal links, such as preconditions. They believe this to be "advantageous for plot generation because plans provide explanations for the behaviour of the characters" (Brenner, 2010). The plot graph thus represents the events that happen in the story in chronological order. The method they present, Continual Multiagent Planning (CMP), is a distributed algorithm. It describes planning by multiple different agents, who have varying goals and beliefs. Given the individual agents' plans may include other agents (such as asking for locations for instance), it also models planning for multiple agents. For storytelling, this means that CMP allows for both character-centric and author-centric plot generation.

Over the course of execution, agents switch between planning, plan execution, monitoring, plan adaptation and communication. Characters are not always willing to adopt each other's plans. Once it has found plan, an agent will enter execution phase. It will then execute the first action in its plan, given its pre-conditions are fulfilled and then monitor the results of the

executed action. Using an implementation of MAPL agent simulation called MAPSIM, they are able to create automatically generated stories. Given the MAPL domain and individual goals and belief for each character and an initial world state, the system could produce a world graph of event sequence in which character action, sensing and communication interleaves naturally. This seamless transition between these types of actions allows for the characters to change their plan when new information is provided that would render their previous plan impossible or not optimal. This also allows, when an agent does not have enough information to accomplish their goal, to create a sub-goal that would make the main one feasible. For instance, if an agent does not know the location of the item it seeks, it would make its current goal to ask for information and once it has been provided with the information, update its plan to go get said item. MAPL has a "currently" attribute allowing it to provide temporary or context dependant motivation and goals for agents. Characters need to commit to goal/desire in order to actively pursue it (commitment precondition). This means that for agent b to do agent a's desire, agent a needs to persuade agent b first.

One of the problems with classical planning algorithm for storytelling is that although they are able to follow the initial intent of the human author, they often lack the ability to make characters believable, an essential element for storytelling. On the other hand, while treating each character as an autonomous agent makes it possible for believable character interaction to emerge, it provides no guarantee the authorial intent will be kept intact. Attempts have been made for intent based planning (Riedl and Young, 2010) but the results took very long to generate, making them ill-suited for real-time use. Teutenberg and Porteous (2013) thus developed a system for intentional multi-agent planning with relevant action they called IMPRACTical in order to counter these shortcomings. They suggest a character based system,

meaning the narrative relies on independent planning agents. Adding external guidance, in the form of a director agent, can help steer the set of possible plan hierarchy in the right direction. The IMPRACTical system is "intentional in that it performs global heuristics search, but is not monolithic as it delegates intent to multiple planning agents". By keeping the intentionality of each agent separated from the global search procedure for the final sequence of action they improve the systems' efficiency while satisficing the authorial goals and narrative goal of having intentional character. The criteria here for intentionality is that it is required that the narrative features causal links from every point (or action) in the plot graph to an intent of the relevant character and that characters must have achieved their intent before executing the action. Their proposed solution to the problem is thus to delegate the task of reasoning about intentions and relevant actions to the agents while a single narrative planner generates narratives which satisfy the selected intentional actions.

They have identified three characteristics of intentional plans that contribute to character intent. In situations where characters cannot by themselves have their intentions fulfilled, the use of co-operating decisions can allow the characters to reach their goals and thus fulfill their intentions. Furthermore, by predicting other character's action and using them as justification in the current plan, the current character seems more intentional as it includes another character's plans in its own. The last, which they call "chains of command", refers to planning several steps ahead while including other character's actions.

The concept of this engine is to delegate the whole reasoning to a group of agents, the characters in the story to be written. Each of these agents will in turn, given a world state, return the sequence of action that it has found to be relevant given its intentions, as opposed to simply finding actions according to a single plan to which they are committed. By giving all intentional

actions, this creates a very large yet intentional search space. This method of reducing the search space only to those actions deemed intentional is in contrast to expanding the space to find product of intent and action. In order for the agent to determine which of the possible actions is relevant, it must be provided with sufficient information, such as which actions are available to him and the current world state. This means that the agent has all possible facts, the world state with all facts that are currently true, all agents, intents and actions.

Agents are going to be cooperating if they have the same intent. This ensures coherence in decision of helping each other achieve an otherwise unachievable goal and makes sure two agents that share mutually exclusive goals will not be cooperative, in keeping with the intentionality and the suspension of disbelief.

The characters have full knowledge of the world state, including other characters. This allows other agents to use one agent's action as a precondition for future actions, which is necessary for the three types of behaviour for intentionality. It allows agents to be able to reason about other agent's plans and to reason about other agent's intentions in an explicit manner.

Lastly, in order to permit the agents to produce plans in the 'chain of command', the planner uses the command given to another agent, and the resulting plan, by considering it as an action in its own plan.

Their system has shown efficiency at creating stories using the same domain as other state of the art system (Haslum, 2012, Riedl & Young, 2010). Their system, even with a wide branching factor, found its solution in 20.4 seconds. Furthermore, IMPRACTical was able to generate 22 different solutions to the domain, showing its ability to generate new forms of narratives.

All these systems have shown the potential for planning algorithms when it comes to narrative generation and handling of agent's intentionality. However, most of these systems are constrained by human authored content or by the search for an end-state of the system itself by the agents. The simulation based, emergent narrative systems have yet to see an implementation using the player as director agent, as a means of directing the story.

Method

In this project I have created a quest-generation engine called CONAN, implemented in software. The CONAN engine's goal is to produce, given an initial state and domain files with all possible actions, novel and coherent quests for a player audience. Additionally, throughout quest resolution, player actions and other world-state altering events, it is able to produce more context-relevant quests as the simulation goes on, effectively producing countless different quests as the world state changes. In order for this to be possible, it requires as input a world definition composed of locations, non-player characters with pre-defined preferences, monsters and items, laws governing the world (such as "when trees are cut down, there are fewer trees"), what actions are possible, as well the pre-requisites and results of said actions. Each of these items in a specific world simulation will be objects within lists representing locations, characters, monsters or items. Each of the objects are defined within the world state by statements such as location(Castle) pertaining to the castle object, defining it as a location. These specifications will determine what is possible within the world and thus which quests can be created.

Once the input is given, the CONAN engine will accomplish its goal by having the NPCs make relevant and coherent plans to solve their goals in accordance with their preferences, which are provided in their individual domain files. These plans, which will now be referred to as quests, are to be transferred to the player character as requests. For instance, a baker NPC that

ran out of bread might want to make more bread to sell, for which he needs more wheat. That NPC might then ask the player character to get him some wheat from the field in exchange for payment.

The following section will describe in detail each element in the system.

Initial State

The initial state contains statements describing all that exists in the story world. I specifically use the modified Aladdin world that has seen much use in the literature (Teutenberg & Porteous, 2013, Haslum, 2012, Riedl and Young, 2010) for comparison purposes. This means that the following elements are present:

- King Jafar who lives in a Castle
- Aladdin, a knight who has a cooperative attitude towards King Jafar
- Jasmine who also lives in The Castle
- A Genie who is in the location 'Magic Lamp' and unable to get out
- A Dragon who lives in the Mountain, is hostile to agents and guards the 'magic lamp'

One may notice that locations have also been mentioned in the above description. These exist in the world and are interconnected such that one may move to and from the castle and the mountain. And the Magic Lamp is only accessible from the cave.

Each of the agents is provided preferences, which are represented by values weighting the actions, described in the domain section. Examples of this might be 'being free' for the genie, 'keep King Jafar alive' for Aladdin or 'acquire wealth' for King Jafar. These statements might

then all be goals for the characters for which they need to create plans to achieve. These preferences are specifically implemented in the agents as higher or lower costs to each action. Aladdin might then have a lower cost to the 'Defend' action. These will be used by the planning system to find goals and sequences of actions for each agent such that they match the agent's characteristics in order to keep their actions intentional.

For further testing, I also use a second initial world state. The purpose of this is to see the effect of a more complex world with more characters, locations, monsters and objects on the scalability of the system and the difference in possible quests generated by the more complex environment. This second world is set to have the following:

- Agents:
  - Baker
  - King
  - Lumberjack
  - Blacksmith
  - Merchant
  - Guard
  - Daughter
- Locations:
  - The Castle, connected to the village
  - The Village, connected to the castle, the bakery, the shop, the wheat field and the forest
  - The Wheat field, connected to the village.
  - The Cave, connected to the forest.
  - The Bakery, connected to the village

- The Forge, connected to the cave
- The Forest, connected to the village and the cave
- The Shop, connected to the village

As well as several items such as a hammer, wheat, a sword and a magic spell book. The monsters (the troll, the wolves and the slimes) will also be considered as items for implementation purposes. Both initial states, as they appear in the program, are available in the appendix.

The domain file

The domain files contain the set of possible actions that the characters in the story may use to achieve whatever goal they may have. Following the analysis presented by Doran and Parberry (2011), the agents will have access to all the atomic actions found in his structural analysis of quests. This is so that the generated plans may include all possible actions and therefore offer the greatest variety of quests and closer resemble human created quests. The actions are as follows: DoNothing, Capture, Damage, Defend, Escort, Exchange, Experiment, Explore, Gather, Give, GoTo, Kill, Listen, Read, Repair, Report, Spy, Stealth, Take and Use.

This set of actions covers the set of possible actions that quests require players to perform in human written quests from commercial video games (Doran & Parberry, 2011) and will be evaluated as a set of actions for the current simulation.

Each action will be implemented in the system the following way, using PDDL syntax, which is the standard for planning algorithms (Russel, Norvig, Canny, Malik, & Edward, 1995):

```
(:action move
        :parameters (?p ?to ?from)
        :precondition (and (location ?to) (location ?from) (player ?p) (at ?p ?from))
        :effect (and (at ?p ?to) (not (at ?p ?from)) (increase (total-cost) 2)))
```

Each action thus contains a name, the parameters used (player, destination and current location in the case of the above example), preconditions establishing what conditions must be true in order for the action to be available (such as the destination being a location) and what the effect of said action will have on the world once it has been performed (such as the location of the player is now the destination). Additionally, actions will be weighted in accordance with agent preferences, with actions incompatible with the agent's preference having higher cost than those in line with the preferences. This is represented with the 'increase (total-cost)' part of the effect, with the number representing the agent's preference for this specific action. All action costs have a base cost of 2. This means that an action such as Kill will have a path cost higher (raise to a cost of 3) for agents such as the baker than the knight, for instance. These mappings of actions to preferences of the agents will guarantee that the agent's choice of actions will remain coherent with its personality and will protect suspension of disbelief. The agents have the following action preferences:

Table 1: Aladdin World character preference

Character	Preference
Aladdin	["+kill","-exchange","-use","+escort"]
Dragon	["-damage","-take","-report","+escort","-defend"]
Genie	["-kill", "-exchange", "-defend", "-read"]
Jasmine	['+kill', '-spy', '-take', '-stealth']
Jafar	["-kill", "-spy", "-take", "-stealth", "move"]

In the above table, actions preceded by a '–' have a lower cost (and are therefore preferred) and those with a + sign have a higher cost, making them less desirable. Each character has been given four or five actions with differing costs, though any number of actions can be modified in this way in order to change the agent's preference. One can see that, in doing so, it is

possible to steer an agent's preference for certain actions and at the same time for certain types of quests. For instance, notice how Jafar has a preference for the actions 'kill', 'spy', 'take', 'stealth', and 'move'. This will cause him to prefer plans that involve sneaky actions, murder and stealing. In the same way, one can change the preferences of an agent to make them prefer defending, trading, or any personality wanted. The exact agent preferences for the large world can be found in the Appendix.

Goal Generation

The goals themselves take the form of sets of statements that must become true in the world state. These statements are a combination of predicates, such as 'has' or 'defended', and an object, such as an agent, an item, or a location. For example one statement in a set might be (has baker wheat), meaning that in order for the goal to be accomplished, that statement has to become true. In the current implementation of the program the possible predicates are as follow, with ?i being information, ?l a location, ?o an object, ?c a character, ?p the player:

```
predicates = [
      "(has ?cl ?i)",
      "(has ?p ?o)",
      "(has ?c ?o)",
      "(cooperative ?c)",
      "(at ?c ?l)",
      "(character ?c)",
      "(captive ?p ?c)",
      "(damaged ?i)",
      "(defended ?c)",
      "(defended ?i)",
      "(sneaky ?p)",
      "(dead ?m)",
      "(experimented ?i)",
      "(explored ?l)",
      "(used ?i)"]
```

These define who or what can have the predicate describing a certain situation. Therefore, the player can only make herself ‘sneaky’ as the ‘sneaky’ predicate can only be attributed to ?p, the player.

The engine uses two algorithms to choose goals in order to compare them. The first one will randomly select goals for each NPC to use as a baseline comparison against the preference-based goals. It does so by choosing a number of random predicates from the above list equal to a user-defined number. It then cycles through the predicates and assigns a random legal item in place of the ‘?’ object.

The second proposed algorithm will, for each agent, select a number of random goals in the same method as was described above and rank them according to the current agent’s preferences and only keep the one best fitting of the agent’s preference, that is, the one with the best score. This is done by taking the list of goal states for each agent and finding a plan to reach said goal. The average cost of the actions in the plan is calculated, giving an idea of how well the plan fits with the current character’s preferences. Given that the cost for actions in line with the preference of an agent is 1, the closer the average is to 1 the more the plan is in line with the character. This is repeated for each goal state and the one yielding the plan with the lowest cost is kept. The engine creates 4 random goal states to be evaluated in this fashion for each character by default. A higher number means the system is more likely to find a better goal but also means it is more taxing in terms of computation as it must create a plan for each said goal. The spline curve in Figure 1 shows how the mean of action costs within a plan changes with the number of goal states attempted. This second method ensures the goals will more likely be compatible with the agents’ preferences, thus preserving the illusion of intentionality.

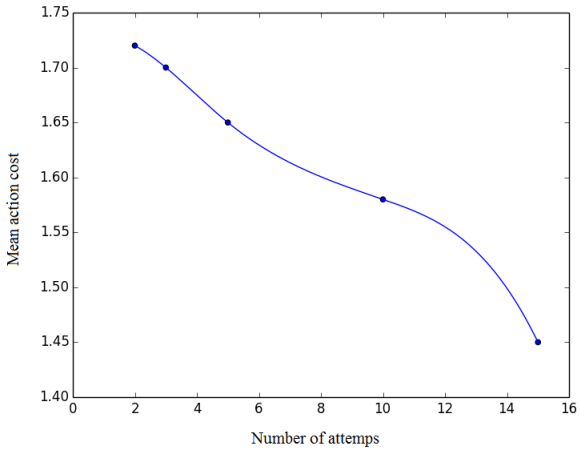

Figure 1: Mean of action cost by number of goal states attempted

Simulation

The simulation itself runs in a turn-based manner. There are two turns: the NPC turn and the PC turn. This means that the engine, after taking in the initial state and domain files with the actions, will solve each agent's planning problem and present the player with each plan. This is the NPC turn, where each of the agents designs their plans and gives it to the player. Before it is given to the player, a distribution of the quest motivations for the generated quests will be computed for evaluation purposes. In future version, before selecting the quest to be given to the player, it will be compared with the distribution and will only be used if it has not been overused. This will use a player model to determine player preference and flatten the quest motivation distribution, only letting the ones the current player prefers be presented in higher frequency. This is in order to prevent the quests presented to the player from being repetitive. A translation module then transforms the plans from the series of action names to a readable form for the player to be able to understand.

Next is the player turn, where the player may perform an action that impacts the game world. It is the opportunity for the player to interact with the NPC—that is, to help them with the

quest they have transferred onto him. Once the action is taken, the world states is updated and the system cycles back to the NPC turn, checking for new plans in light of the changes that have occurred in the world. An example case scenario of a simulation is shown in appendix B.

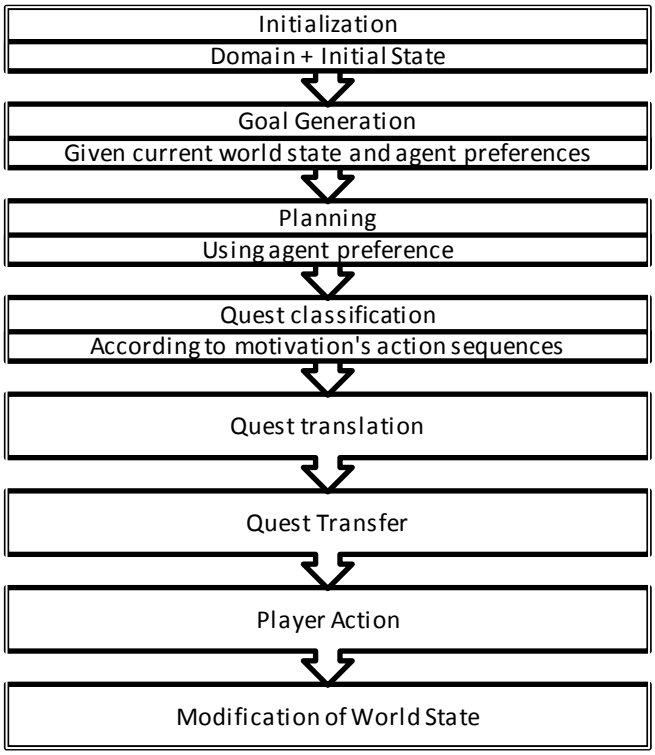

Figure 2: Flow of operation during the course of a simulation. The operation cycles from 'Goal Generation' onwards.

Quests

As mentioned previously, the CONAN engine presents the player with various quests at each time step. "Quest" is here defined as a series of steps that must be taken in order to solve some problem, as presented by the agents to the player as a request. The quests are formally and structurally defined as follows.

In their analysis of more than 3000 human authored quests designed for commercial video games, Doran and Parberry (2011) classified quests into various categories. They observed that each quest could be classified into one of 9 different categories according to its underlying

motivation. Motivation is, according to Doran and Parberry (2011), essential for ensuring that quests appear intentional and appropriate rather than randomly generated. The nine categories of motivation are as follows: knowledge, comfort, reputation, serenity, protection, conquest, wealth, ability, and equipment. The above example pertaining the baker's wheat would fall under the "Wealth" category. Table 2 describes examples of quests that the CONAN engine could generate for each of the different categories. In order to create a system that is able to create a wide variety of realistic quests, my goal was for the engine to create quests from each of these categories.

The example quests given below assume an initial world state as described in the second, more complex state in the Initial state section.

Table 2: Categories of Quest motivation

Motivation	Example of quest
Knowledge	"Find the location of the king's stolen treasure"
Comfort	"Get rid of the wolves in the forest that are preventing the lumberjack from getting wood."
Reputation	"Get granite and build a statue of me in the town square."
Serenity	"Rescue the daughter of the baker that was taken by a troll."
Protection	"Go kill the troll that has been traumatizing the village"
Conquest	"Go kill my enemies."
Wealth	"Go get some wood for the lumberjack to sell."
Ability	"Find me the ancient spell book."
Equipment	"Repair the lumberjacks' axe."

Furthermore, given the actions that exist within the current software and its implementation, the quests will fall within Aarseth's (2005) first and last categories, place oriented and objective oriented, given that they are to be NPC's plans to attain an objective

within the game world, in different areas. The second category, time-oriented, cannot exist in the current version as there is no implemented concept of duration.

Using NPC preferences and a varied action set results in a variety of possible quests proportional to the variety of character motivation, action, and objects in the world implemented at initiation, spanning possibly the entire range of possible categories. These are the building elements of the quests and define the type of quests that will be generated by giving it its concrete goal. Smaller, simpler initial states will create fewer, less varied quests while more complex worlds will create quests spanning all quest structures.

The solution to each NPC's current problem or objective will be processed individually by a planning agent. In order to do that, each NPC is instantiated as a planning problem with its own set of constraints and goals and its own domain. As Teutenberg and Porteous (2013) have noted, although authorial intent suffers from such distributed processing, characters are more realistic, which is important for character believability, which had been noted as providing meaningful interaction (Aylett, Dias & Paiva, 2006; Teutenberg & Porteous, 2013).

The quests, as mentioned previously, will be the plans the system constructs for each agent. This means that for each agent, during the NPC phase, the system will take the current state of the world and the constraints imposed by the specific agent and find the least expensive path, in terms both of number of actions and of weight relative to its preference, to its current goal. For instance, in a world defined as follows:

NPC(Sebastien):
  Preference(Baking)
  Goal(Not(Hungry) AND Rich)
Has(Sebastien, Bakery)
Location(Bakery)
Location(Wheat Field)
Item(Wheat)
Has(Wheat Field, Wheat)

Figure 3: Sample initial world state

The planning engine might come up with the following plan:

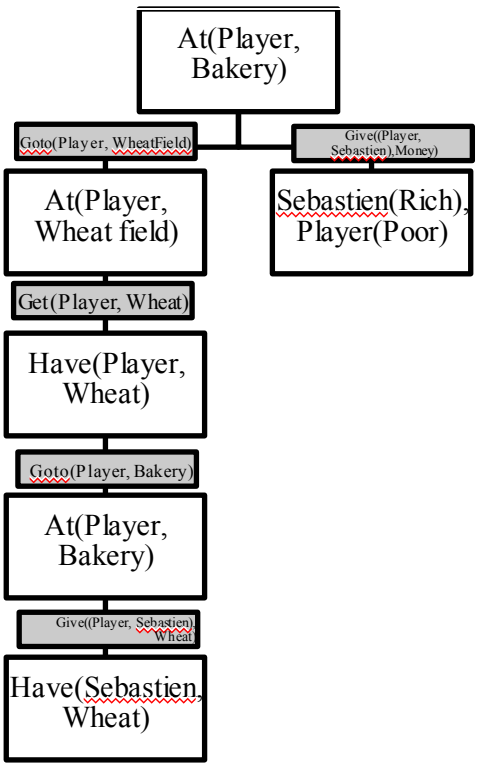

Figure 4: Sample plan for solving the need for wheat from the baker. Both branches show possible plans solving the goals of the NPC.

Once the agent knows the plan is possible, achieves current goal, and has the lowest cost (therefore meaning it is in line with its preferences), it is accepted as the current plan and is translated for the player to be able to read.

Classifier

In order to determine the motivation category of each of the created quests, the engine has a build-in classification module. This module uses the quest structure from the classification of Doran and Parberry (2010) to compare it against the ones it created. In their work, based on human written quests from commercial video games, they have found that all quests can be associated with one of nine broad categories, the motivations, which are shown in Table 2. This broadest classification represents different types of goals the NPC wanted achieved through the quest itself. In order to achieve the goal of the NPC, the quest employed different strategies, which are specific to said motivation, each motivation having between two and seven different strategies specific to itself, the exhaustive list of which can be found in Doran and Parberry (2010). The strategies themselves, being a way to reach the quests' goal, are high level representations of a specific sequence of action that must be performed in order to attain the goal. For example, the quest motivation "Knowledge" has four associated strategies, which are unique to it. These are "Deliver item for study", "Spy", "Interview NPC", and "Use and item in the field". Each of these strategies is described by a sequence of action, as shown in the table below.

Table 3 : Strategy and sequence of actions for the Knowledge motivation

Motivation	Strategy	Sequence of Action
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>

The classifier, in order to determine to which broad category, or motivation, the created quest belongs to, creates a fuzzy membership based classification where each strategy belongs (0 to 1) to a quest. To do this, the quests are segmented into pairs of actions, for which the classifier searches through every strategy, for each of the motivations. If a pair is found, its score is augmented. This way, each strategy gets a score and, in turn, the sum of the strategies' score gives the motivation its own score. The motivation with the highest score is determined to be the motivation for the quest. This can be described by the following:

$$\sum_{i}^{N} \frac{1}{L_i \times N}$$

Where N is the number of strategies in that motivation, and L_i is the length of a given strategy, specifically the number of pairs of actions.

This method of classifying quests has the advantage of allowing the classifier to detect strategies if they are not complete, as long as the sequence of action is closer to a given strategy than another. One inherent flaw to this process comes from the classification itself. Since some strategies are comprised of only one action, such as the 'Ability' motivation with its 'Use', and 'Damage' strategies, the classifier will consider a sequence of actions such as ('Move' 'Damage') to be both from ability because of the 'Damage' and from the 'Serenity' motivation, as one of its strategies is ('Move' 'Damage'). This causes the motivation with single action strategies to be over represented, such as 'Ability'. Furthermore, the classifier also does not account for possible sub quests or imbedded quests. For instance, if the quest was to destroy something with an axe but the character first had to get said axe, the classifier would count this as only one quest with one motivation instead of a quest with an imbedded sub quest.

In order to test the classifier itself, it was compared to quests classified by a human. I took the first 50 quests that were output by the below-described large world test and classified them by hand into each of the 9 motivation categories. I then used the same 50 quests and used the classifier to divide them into the 9 motivations. Out of the 50 motivation assignments, 14 were not used because the classifier could not classify them, leaving 36, on which the classifier and I agreed on 47.2%. Although the agreement is not very high, another test, assigning random motivations to the same quests a thousand times found that the random classifier would have an average agreement with my own classification of 11%, showing that the current classifier is more in tune with human classification than random.

The difference in classification between the module and I can be explained in part by the inherent nature of the quest structure found in Doran and Parberry (2010). Although the motivations have specific strategies, some of them are very similar. For instance, the Knowledge motivation has the "deliver item for study" and the Equipment motivation has the "deliver supplies" strategy. Both these strategy have for sequence of action "<get><goto><give>". Such similarities lead to ambiguity in decision of which motivation should be assigned to a given quest.

Planning algorithm

Different solutions have been used by others as described in the overview of the literature and seem to fulfill the task as per their respective system's requirement. The CONAN engine need not create immensely complex plans (or quests) as the complexity of the narrative is theorized to emerge from the variety of interactions the player has with the world's agents. Furthermore, given that the planning is done by each agent for themselves, the intentionality of

the plan can be dealt with directly by the planner during planning, where the preferences provide differing costs to different actions. More complex distributed algorithms such as MAPL (Brenner, 2010) is unnecessarily complex for the proposed engine and the handling of intentionality by the global planner (Riedl and Young, 2010, Teutenberg and Porteous, 2013) might not be relevant to the current system, as the use of weighted actions relative to character preferences might be more than enough to guarantee intentionality.

The CONAN system uses Fast Downward (Helmert, 2006), as distributed under the GNU General Public License. It is an implementation of classical planning system based on heuristic search. It is encoded in PDDL 2.2, which is a standardized format for planning, and supports features such as ADL conditions and effect. This had the advantage of allowing numerical values to be used in the domain file, which I use here for weight of each action to be added in the plan weight. ADL also provides support for negative statements in preconditions. The planner supports many heuristics and I used A* in the current implementation. It takes in a standard PDDL format problem, translates it into multi-valued planning task and uses a hierarchical decomposition of the planning task to compute the heuristic function. I used this implementation of the planning engine as it supports negative clauses and numerical values. Furthermore, it has proven itself a very efficient implementation (Helmert, 2006).

Evaluation

The created plans consist of sequences of actions leading the current state to the goal state. Each action sequence is compared against Doran and Parberry's (2010) sequences of actions that belong to strategies underlying each of the motivations. For each strategy found in the plan, its associated motivation will be given a score and the motivation with the highest score

will be determined as the motivation for the quest. The engine will be determined to be exhaustive in its breadth of possible quests if it is able to cover all the nine motivations that were found underlying all the human authored quests in the analysis by Doran and Parberry (2010).

Large World Test

In order to verify the breadth of possible quests, the simulation needed to create many quests. A world was used to create quests through 1000 iterations. Considering the large world being the more complex of the two, it was used for this test. The simulation was altered very slightly in order to make the processing of a large number of plans possible. The PC phase was not used and, while creating new goals, only random goals were chosen to cut on processing time, as this meant creating 7000 quests. In each iteration, a plan was created by every agent without modifying the initial world state. The reason for this was twofold. Firstly, the goal for this test was only to create as many quests as possible in order to assess the breadth of created quests, meaning the effect of a changing world state was not under evaluation. This means that it allowed us to speed up processing by skipping both the plan update upon world state change and the modification on world state. Secondly, this allowed the breadth of quests to be assessed from a single state. Considering that different states create more variety, the amount of different types of quests from a single state was assessed. One can estimate that the variety obtained from a single state would be much smaller than the possible diversity of quests from multiple different world state. Furthermore, most commercial games have a set initial state that does not change from the time it is launched.

Every iteration, each agent created a single quest. The quests were then classified using the classifier module into the 9 motivations found in Doran and Parberry (2011) depending on the strategies found in the plans. Table 4 shows the resulting motivation distribution.

Table 4: Number of quests in each motivation

Motivation	Count	%
Ability	1099	0.176631
Comfort	392	0.063002
Conquest	165	0.026519
Equip	227	0.036483
Knowledge	281	0.045162
Protection	456	0.073288
Reputation	975	0.156702
Serenity	505	0.081164
Wealth	424	0.068145
Not Found	1698	0.272903
Total	6222	1

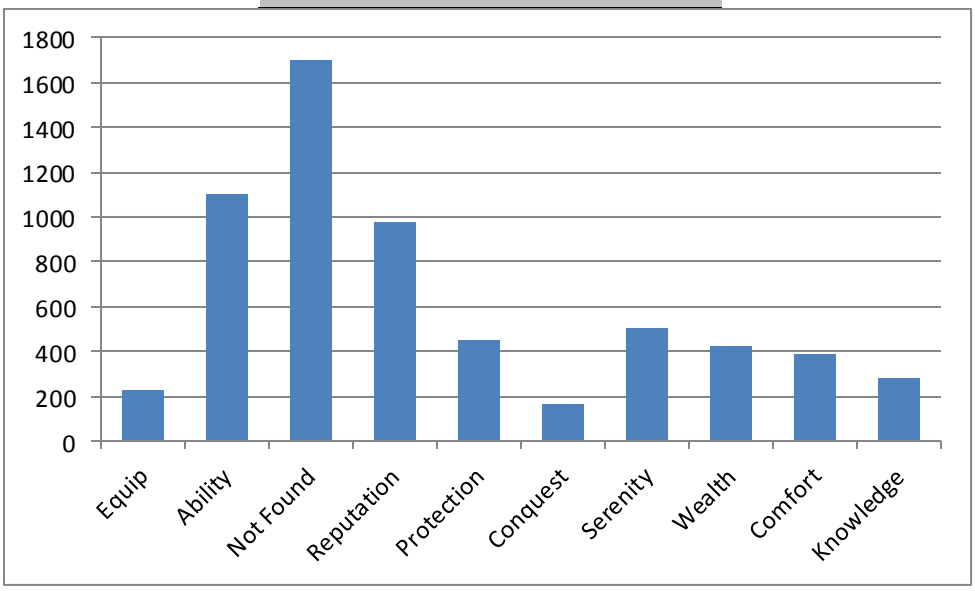

Figure 5: Distribution of quest motivation

As one may see from Figure 6, all the motivations were found in the Large World Test. Using a single initial state, the agents were able to create quests spanning all of the broad categories describing all human-writing quests. This confirms that the engine is indeed capable

of creating a wide variety of quests to present to the player audience. The engine's distribution (Figure 6) appears to be different from the distribution of human-generated quests (Figure 7) as described in Doran and Parberry (2011). One may speculate that a possible explanation for this difference might be attributed to human author preference and current popularity in the game market in the case of the human authored quests, where the engine does not have such bias towards a specific quest motivation.

Table 4: Quest Motivation found in human written quests

Motivation	Count	%
Ability	8	0.01062
Comfort	12	0.01594
Conquest	152	0.201859
Equip	139	0.184595
Knowledge	138	0.183267
Protection	137	0.181939
Reputation	49	0.065073
Serenity	103	0.136786
Wealth	15	0.01992
Total	753	1

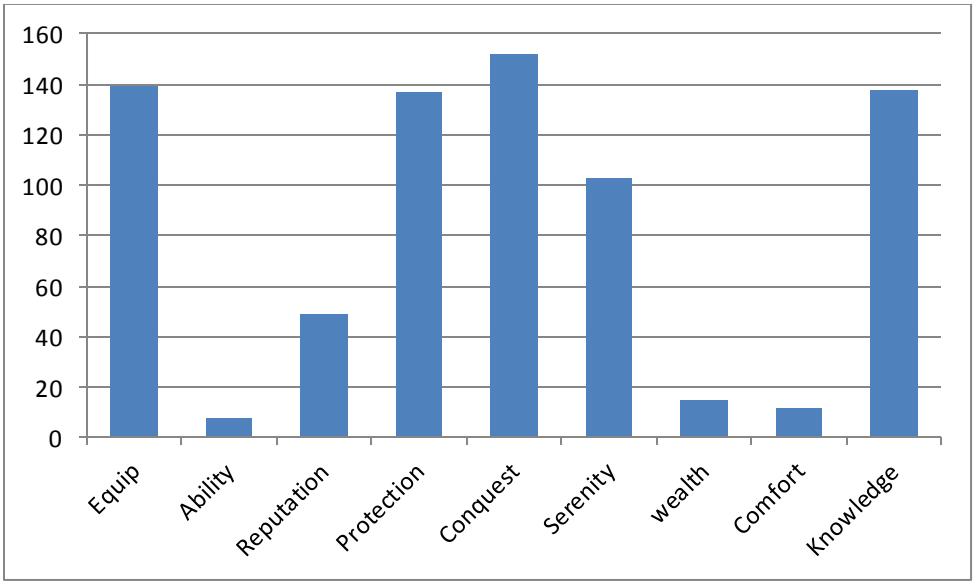

Figure 6 : Distribution of motivation in human written quests

Many quests fall under the 'NotFound' category. These are quests that were either impossible to execute or already complete within the world. Given that for this test, the goals were chosen randomly, it was not impossible for the engine to choose goals that already exist within the world, such as (alive baker). These goals could not produce quests and were therefore placed under the 'NotFound' category. Similarly, quests for which a successful plan could not be found within 5 minutes of searching were also dismissed as 'NotFound'. Any quest with the 'NotFound' category would not be transferred to the player, being considered an unplayable quest. The goal generation did not automatically remove goals statements that already existed in the world state for two reasons. Firstly, although it would be beneficial for smooth gameplay, the existence of already resolved quests does not impact the capability of the engine to create other quests. Secondly, these already resolved goals could be used, in future implementation, for quests creation as well. Indeed, although the current implementation does not have this feature, goals that are already reached could be considered states that must not be changed. For instance, if the guard was to choose the goal (defended village), and such a statement already existed in the world state, once could interpret this as a quest to keep the village defended. Future implementation could use this to create even more different quests, possibly implementing the time-oriented quests from Aarseth (2005). As for impossible quests, these are quests where the engine tried and failed to create a plan given the goals. This could be for different reasons such as conflicting goals.

Aladdin World Test

The second test used the modified Aladdin world for its simulation. This world is much simpler, having fewer facts and objects than the large world. The same test from the Large World Test was performed, using only NPC phase, random goals and 1000 iterations creating plans for

each of the 5 agents. Given its much simpler nature, the world did not produce the same variety of quests as the Large World.

Table 5: Quest motivation in Aladdin World

Motivation	Count	%
Ability	455	0.16055
Comfort	79	0.027876
Conquest	0	0
Equip	0	0
Knowledge	99	0.034933
Protection	0	0
Reputation	0	0
Serenity	937	0.330628
Wealth	0	0
NotFound	1264	0.446013
Total	2834	1

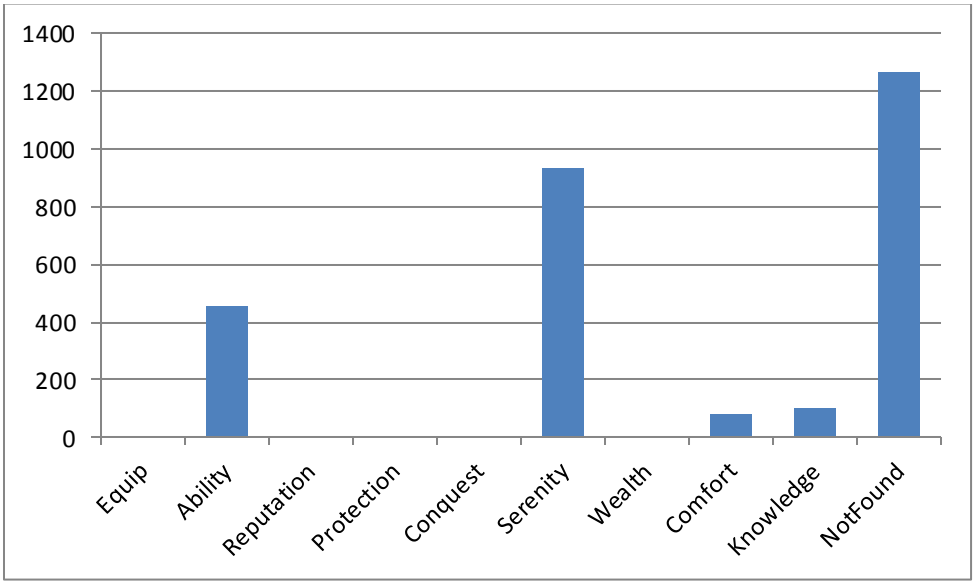

Figure 7: Distribution of quest motivation in Aladdin World

Conclusion

In conclusion, the CONAN engine is indeed capable of creating quests that resemble human written quests in a structural manner. Although the tests showed a different motivation distribution for the sets of quests found from the human written quests, the engine can indeed create quests that cover the entire set of categories. It has also shown that although worlds with few objects could not produce all possible quest motivation, the CONAN engine can produce them given enough information, supporting the claim that the complexity of the produced quests depends on the complexity of the world given as input. The more complex the given world, the more complex the quests presented to the player audience will be. Such an engine, based on a simulation approach to narrative generation, using intentional agents to provide a player, that acts as a director agent and player model, with believable interaction could produce the emergence of an interesting and coherent piece of narrative. The interaction with the agents, referred to here as quests, could be the building blocks of the story that unfolds as the player performs these quests for the various agents provided by the system. Furthermore, since the generative nature of each of these block depends of the world state at the time of generation, the set of possible quests at any moment is constrained only by the current world, which changes as the simulation goes on, effectively creating a constantly changing set of possible quests.

Such an engine, effectively capable of autonomously creating infinity of quests would be valuable to an industry that is growing in demand for complexity and which uses human artists and authors to create such content. The possibility to offload this task, partially or totally onto an autonomous system would make the creation of such games much cheaper and create a very high replay value as the gameplay would be different every time the game is played by its intended audience.

The current work differs significantly from other similar systems by its reliance on emergence and player interactions. State of the art systems such as Teutenberg and Porteous' IMPRACTical (2013), seek to create stories through intentional planning by multiple agents and a single narrative planner that generates the narrative story. Brenner's MAPL based system MAPSIM (2010) uses continuous multi-agent planning in order to write a story with its different agent's goals and intentions, and Riedl, Thue and Bulitko (2008) use a centralized planner to adapt plotlines in order to create new stories. Lastly, Doran and Parberry (2010) use structural rules to create quests from their analysis. The CONAN system's main difference comes from its simulation based, emergent approach and its iterative process where story is generated through each iteration of the player's actions. I forgo the use of story planners or director agents in favor of an interactive approach where the player dictates, through their decision during the simulation, how the story ought to evolve. This approach has yet to be investigated in depth and can provide a different approach to generation of stories in games, which would be less involved in terms of human authoring and thus more efficient than the traditional human written stories.

Future work should investigate the interaction aspect of the proposed theory. Specifically, one may look at subjective experience of subject reporting on both their interactions with the characters in the world but also of their experience of any single simulation's story as they progress through the quest and interact with the agents in the world, giving information on the believability of the characters.

Furthermore, studies should investigate the current engine capabilities for emergence of story. The current study does not assess believability of the agents nor does it investigate the effect of the simulation when presented to, and interacting with a human player. This interaction is theorized to create, in the mind of the audience, a story that is personal and different for every

player, every time. Furthermore, this study does not investigate qualitative and subjective assessment of interestingness or suspension of disbelief. Studies looking to further this engine should implement it into a proper game and test it using participants.

References

Aarseth, E. (2005). From Hunt the Wumpus to EverQuest: introduction to quest theory. *Entertainment Computing-ICEC 2005* (pp. 496-506). Springer Berlin Heidelberg.

Ashmore, C., & Nitsche, M. (2007, September). The quest in a generated world. *Proc. 2007 Digital Games Research Assoc.(DiGRA) Conference: Situated Play* (pp. 503-509).

Aylett, R., Dias, J., & Paiva, A. (2006). An affectively-driven planner for synthetic characters. *Proc. of 16th Int. Conf. on Automated Planning and Scheduling* (ICAPS-06), 2006.

Bartle, R. (2003). Designing Virtual Worlds. New Riders Games.

Brenner, M. (2010). Creating Dynamic Story Plots with Continual Multiagent Planning. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. pp.1517-1522

Brenner, M., & Nebel, B. (2009). Continual planning and acting in dynamic multiagent environments. *Journal of Autonomous Agents and Multiagent Systems*, vol. 19(3)

Chatman, S. (1993). *Reading Narrative Fiction*. Macmillan Publishing Company, New York.

Dennett, D. (1989). The Intentional Stance. MIT Press, Cambridge, MA

DesJardins, M., Durfee, E., Ortiz, J. & Wolverton, M. (1999). A survey of research in distributed, continual planning. *The AI Magazine, vol 40(4)*

Doran, J., & Parberry, I. (2011, June). A prototype quest generator based on a structural analysis of quests from four MMORPGs. *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games* (p. 1-8).

Edwards, M. (2011). Algorithmic composition: computational thinking in music. *Communications of the ACM*, *54*(7), 58-67.

GNU General Public License, version 3. Free Software Foundation. http://www.gnu.org/copyleft/gpl.html. Retrieved on Nov 11, 2014.

Greuter, S., Parker, J., Stewart, N., & Leach, G. (2003, February). Real-time procedural generation of pseudo infinite cities. *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (pp. 87-95). ACM.

Hartsook, K., Zook, A., Das, S., & Riedl, M. O. (2011, August). Toward supporting stories with procedurally generated game worlds. *Computational Intelligence and Games (CIG), 2011 IEEE Conference on* (pp. 297-304). IEEE.

Haslum, P. (2012). Narrative Planning: Compilations to Classical Planning. *Journal of Artificial Intelligence Research 44, pp. 383-395*

Hastings, E. J., Guha, R. K., & Stanley, K. O. (2009, September). Evolving content in the galactic arms race video game. *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on* (pp. 241-248). IEEE.

Helmert, M., (2009). The Fast Downward Planning System. *Journal of Artificial Intelligence26, pp. 191-246*

Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural content generation for games: a survey. *ACM Transactions on Multimedia Computing, Communications, and Applications*, *9*(1), 1.

Hong, Y., & Riedl, M., O. (2012). A Sequential Recommendation Approach for Interactive Personalized Story Generation. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems,* pp. 4-8

Kelly, G., & McCabe, H. (2006). A survey of procedural techniques for city generation. *ITB Journal*, *14*, 87-130.

Laird, J. E., van Lent, M., (2001). Human-Level AI's Killer Application Interactive Computer Games. *In AI Magazine, vol. 22(2), pp. 15-25*

Lefebvre, S., & Neyret, F. (2003, April). Pattern based procedural textures. *Proceedings of the 2003 symposium on Interactive 3D graphics* (pp. 203-212).

Li, B., Riedl, M. O. (2010). An Offline Planning Approach to Game Plotline Adaptation. *Proceedings of the 6th Conference on Artificial Intelligence for Interactive Digital Entertainment, Palo Alto, California*

Magerko, B., Laird, J., Assanie, M., & Stokes, A. K. D. (2004). AI Characters and Directors for Interactive Computer Games. *Proceedings of AAAI*, 2004.

Martin, A., Lim, A., Colton, S., & Browne, C. (2010). Evolving 3d buildings for the prototype video game subversion. *Applications of Evolutionary Computation* (pp. 111-120). Springer Berlin Heidelberg.

Orkin, J. (2004, June). Agent Architecture Considerations for Real-Time Planning in Games. *AIIDE* (pp. 105-110).

Prince, G. (1987). *A Dictionary of Narratology*. University of Nebraska Press, Lincoln.

Re, A., Abad, F., Camahort, E., & Juan, M. C. (2009). Tools for procedural generation of plants in virtual scenes. *ICCS 2009: International Conference on Computational Science*. Springer-Verlag, Berlin, Heidelberg, pp. 801–810.

Riedl, M., Thue, D., & Bulitko, V. (2011). Game AI as storytelling. In *Artificial Intelligence for Computer Games* (pp. 125-150). Springer New York.

Riedl, M., O., & Young, R., M. (2010). Narrative Planning: Balancing Plot and Character. *Journal of Artificial Intelligence Research 39(1)*, pp. 217-268

Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., & Edwards, D. D. (1995).*Artificial intelligence: a modern approach* (Vol. 2). Englewood Cliffs: Prentice hall.

Sims, K. (1991). Artificial evolution for computer graphics. *SIGGRAPH. ACM,* pp. 319–328

Sun, J., Yu, X., Baciu, G., & Green, M. (2002). Template-based generation of road networks for virtual city modeling. *Proceedings of the ACM symposium on Virtual reality software and technology* - p. 33-40.

Teutenberg, J., & Porteous, J. (2013). Efficient Intent-Based Narrative Generation Using Multiple Planning Agents. *Proceedings of the 12th International Conference on Autonomous Agent and Multiagent Systems (eds.) May, 6-10, 2013, Sait Paul, Minnesota, USA*.

Togelius, J., Preuss, M., & Yannakakis, G. N. (2010, June). Towards multiobjective procedural map generation. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* (p. 3). ACM.

Tosca, S. (2003, March). The quest problem in computer games. *Technologies for Interactive Digital Storytelling and Entertainment (TIDSE) conference, in Darmstadt*.

Whitehead, J. (2010, June). Toward procedural decorative ornamentation in games. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* (p. 9). ACM.

Yu, H., & Riedl, M. O. (2012, June). A sequential recommendation approach for interactive personalized story generation. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1* (pp. 71-78). International Foundation for Autonomous Agents and Multiagent Systems.

Appendix A

World states

Large World	Aladdin World
facts = set(["(character baker)", "(character blacksmith)", "(character king)", "(character lumberjack)", "(character merchant)", "(character guard)", "(character daughter)", "(monster troll)", "(monster wolf)", "(monster slime)", "(location forge)", "(location bakery)", "(location cave)", "(location field)", "(location castle)", "(location village)", "(location forest)", "(location shop)", "(item ore)", "(item wheat)", "(item hammer)", "(item sword)", "(item spellbook)", "(weapon sword)", "(weapon hammer)", "(has field wheat)", "(has cave ore)", "(has blacksmith ore)", "(has troll hammer)", "(has guard sword)", "(at baker bakery)", "(at blacksmith forge)", "(at troll cave)", "(at lumberjack village)", "(at you bakery)", "(at king castle)",	facts = set(["(character Jafar)", "(character Aladdin)", "(character Jasmine)", "(monster Dragon)", "(monster Genie)", "(location castle)", "(location mountain)", "(location lamp)", "(item lamp)", "(information secret)", "(player you)", "(has Dragon lamp)", "(has Jafar secret)", "(at Dragon mountain)", "(at Jafar castle)", "(at Jasmine castle)", "(at you castle)", "(at Genie lamp)", "(at Aladdin castle)", "(adjacent mountain lamp)", "(adjacent lamp mountain)", "(adjacent mountain castle)", "(adjacent castle mountain)", "(= (total-cost) 0)"])

"(at shop merchant)", "(at guard village)", "(at daughter cave)", "(at wolf forest)", "(at slime field)", "(cooperative blacksmith)", "(player you)", "(captive troll daughter)", "(adjacent castle village)", "(adjacent village field)", "(adjacent village bakery)", "(adjacent village forge)", "(adjacent village forest)", "(adjacent village shop)", "(adjacent field cave)", "(= (total-cost) 0)"])	

Large World Character Preferences

Agent	**Preference**
Baker	["+kill","-exchange","-use","+escort"]
Blacksmith	["+escort", "-getfromlocation", "-move"]
Daughter	["-kill", "-spy", "-take", "-stealth","-move"]
Guard	["-damage","-take","-report","+escort","-defend"]
King	["-kill","-spy","-read","-move"]
Lumberjack	["-getfromlocation","-escort","-giveto","+move", "-kill"]
Merchant	["-exchange", "-take", "+explore", "+spy"]

Appendix B

Example case scenario

Initialization

World State	Goal choosing
objects = set(["baker", "king", "lumberjack", "blacksmith", "merchant", "guard", "daughter", "troll", "wolf", "slime", "you", "hammer", "wheat", "sword", "spellbook", "ore", "castle", "village", "cave", "field", "bakery", "forge", "forest", "shop", "fireball", "gel", "pelt", "secret"]) facts = set(["(character baker)", "(character blacksmith)", "(character king)", "(character lumberjack)", "(character merchant)", "(character guard)", "(character daughter)", "(monster troll)", "(monster wolf)", "(monster slime)", "(item pelt)", "(item gel)", "(has wolf pelt)", "(has slime gel)", "(location forge)", "(location bakery)", "(location cave)", "(location field)", "(location castle)", "(location village)", "(location forest)", "(location shop)", "(item ore)", "(item wheat)", "(item hammer)", "(item sword)", "(item spellbook)", "(information fireball)", "(has spellbook fireball)", "(information secret)", "(has king secret)", "(weapon sword)",	{'Serenity': 0, 'Knowledge': 0, 'Reputation': 0, 'Protection': 0, 'Ability': 0, 'Comfort': 0, 'Conquest': 0, 'Equipment': 0, 'Wealth': 0, 'Not found': 0} **baker** (sneaky you) **baker** (defended daughter) **baker** (at king castle) **baker** (cooperative baker) **blacksmith** (and (captive you guard) (experimented pelt)) **blacksmith** (has blacksmith wheat) **blacksmith** (defended sword) **blacksmith** (and (experimented hammer) (defended pelt) (has you secret)) **daughter** (and (experimented sword) (defended spellbook) (at king village)) **daughter** (and (damaged hammer) (dead troll) (used ore)) **daughter** (and (character guard) (damaged spellbook)) **daughter** (has king fireball) **guard** (and (used ore) (captive you lumberjack) (has blacksmith fireball)) **guard** (and (cooperative lumberjack) (defended baker)) **guard** (and (defended ore) (defended merchant) (has merchant pelt))

"(weapon hammer)", "(has field wheat)", "(has cave ore)", "(has blacksmith hammer)", "(has troll hammer)", "(has you sword)", "(at baker bakery)", "(at blacksmith forge)", "(at troll village)", "(at lumberjack village)", "(at you bakery)", "(at king field)", "(at merchant shop)", "(at guard shop)", "(at daughter cave)", "(at wolf village)", "(at slime village)", "(has castle spellbook)", "(cooperative baker)", "(player you)", "(captive troll daughter)", "(adjacent castle village)", "(adjacent village field)", "(adjacent village bakery)", "(adjacent village forge)", "(adjacent village forest)", "(adjacent village shop)", "(adjacent field cave)"	**guard** (and (explored castle) (used spellbook) (explored shop)) **king** (and (has merchant wheat) (defended sword) (explored forest)) **king** (cooperative king) **king** (and (dead troll) (defended gel) (experimented spellbook)) **king** (and (experimented spellbook) (dead troll) (at king field)) **lumberjack** (and (used wheat) (explored field) (experimented hammer)) **lumberjack** (cooperative daughter) **lumberjack** (captive you lumberjack) **lumberjack** (and (character lumberjack) (has you fireball) (damaged wheat)) **merchant** (defended lumberjack) **merchant** (used spellbook) **merchant** (and (experimented hammer) (damaged sword)) **merchant** (and (captive you blacksmith) (captive you blacksmith)) Chosen goals: **Baker:** (defended daughter) **Blacksmith:** (has blacksmith wheat) **Daughter:** (has king fireball) **Guard:** (and (defended ore) (defended merchant) (has merchant pelt)) **King:** (and (experimented spellbook) (dead troll) (at king field)) **Lumberjack:** (cooperative daughter) **Merchant:** (and (experimented hammer) (damaged sword))

First Iteration

World State Modification	Quest plans
	baker (move you cave bakery) (defend you daughter cave) **blacksmith** (move you field bakery) (getfromlocation you field wheat) (move you forge field) (exchange you blacksmith hammer wheat forge) **daughter** (move you castle bakery) (read you castle spellbook fireball) (move you field castle) (report you king fireball field) **guard** (stealth you) (move you village bakery) (take you wolf pelt village) (move you shop village) (giveto you merchant pelt shop) (defend you merchant shop) (move you cave shop) (defend you ore cave) **king** (move you village bakery) (kill you troll hammer village sword) (move you castle village) (getfromlocation you castle spellbook) (experiment you spellbook) Wealth **lumberjack** (escort you baker bakery cave) (giveto you daughter sword cave) **merchant** (damage you sword bakery sword) (move you forge bakery) (exchange you blacksmith hammer sword forge) (experiment you hammer)

Action taken:

(move you castle bakery)

Second iteration:

World State Modification	Quest plans
Statements added: (at you castle) **Statements removed:** (at you bakery)	**baker** (move you cave castle) (defend you daughter cave) **blacksmith** (move you field castle) (getfromlocation you field wheat) (move you forge field) (exchange you blacksmith hammer wheat forge) **daughter** (read you castle spellbook fireball) (move you field castle) (report you king fireball field) **guard** (stealth you) (move you village castle) (take you wolf pelt village) (move you shop village) (giveto you merchant pelt shop) (defend you merchant shop) (move you cave shop) (defend you ore cave) **king** (getfromlocation you castle spellbook) (experiment you spellbook) (move you village castle) (kill you troll hammer village sword) **lumberjack** (explore you castle cave) (giveto you daughter sword cave) **merchant** (damage you sword bakery sword) (move you forge castle) (exchange you blacksmith hammer sword forge) (experiment you hammer)

Action taken:

(read you castle spellbook fireball)

Third Iteration:

World State Modification	Quest plans
Statements added: (has you fireball)	**baker** (move you cave castle) (defend you daughter cave) **blacksmith** (move you field castle) (getfromlocation you field wheat) (move you forge field) (exchange you blacksmith hammer wheat forge) **daughter** (move you field castle) (report you king fireball field) **guard** (stealth you) (move you village castle) (take you wolf pelt village) (move you shop village) (giveto you merchant pelt shop) (defend you merchant shop) (move you cave shop) (defend you ore cave) **king** (getfromlocation you castle spellbook) (experiment you spellbook) (move you village castle) (kill you troll hammer village sword) **lumberjack** (explore you castle cave) (giveto you daughter sword cave) **merchant** (damage you sword bakery sword) (move you forge castle) (exchange you blacksmith hammer sword forge) (experiment you hammer)

Action taken
(move you field castle)

Fourth Iteration

World State Modification	Quest plans
Statements added: (at you field) **Statements removed:** (at you castle)	**baker** (move you cave field) (defend you daughter cave) **blacksmith** (getfromlocation you field wheat) (move you forge field) (exchange you blacksmith hammer wheat forge) **daughter** (report you king fireball field) **guard** (stealth you) (move you village field) (take you wolf pelt village) (move you shop village) (giveto you merchant pelt shop) (defend you merchant shop) (move you cave shop) (defend you ore cave) **king** (move you village field) (kill you troll hammer village sword) (move you castle village) (getfromlocation you castle spellbook) (experiment you spellbook) **lumberjack** (explore you field cave) (giveto you daughter sword cave) **merchant** (damage you sword bakery sword) (move you forge field) (exchange you blacksmith hammer sword forge) (experiment you hammer)

Action taken:

(report you king fireball field)

Fifth Iteration:

World State Modification	Quest plans
Statements added: (has fireball king)	**baker** (move you cave field) (defend you daughter cave) **blacksmith** (getfromlocation you field wheat) (move you forge field) (exchange you blacksmith hammer wheat forge) **daughter** () **guard** (stealth you) (move you village field) (take you wolf pelt village) (move you shop village) (giveto you merchant pelt shop) (defend you merchant shop) (move you cave shop) (defend you ore cave) **king** (move you village field) (kill you troll hammer village sword) (move you castle village) (getfromlocation you castle spellbook) (experiment you spellbook) **lumberjack** (explore you field cave) (giveto you daughter sword cave) **merchant** (damage you sword bakery sword) (move you forge field) (exchange you blacksmith hammer sword forge) (experiment you hammer)