Hierarchical Generation of Dynamic and Nondeterministic Quests in Games

Edirlei Soares de Lima, Bruno Feijó, Antonio L. Furtado
Department of Informatics – Pontifical Catholic University of Rio de Janeiro (PUC-RIO)
Rua Marquês de São Vicente, 225 – Rio de Janeiro – Brazil
{elima, bfeijo, furtado}@inf.puc-rio.br

ABSTRACT
Quests are a fundamental storytelling mechanism used by computer role-playing games to engage players in the game's narrative. Although role-playing games have evolved in many different ways in the last years, their basic narrative structure is still based on static plots manually created by game designers. In this paper, we present a method for the generation of dynamic quests based on hierarchical task decomposition and planning under non-determinism. The proposed approach combines planning, execution, and monitoring to efficiently handle nondeterministic events and support quests with multiple endings that affect the game's narrative and create interactive and dynamic story plots. These plots are directly or indirectly affected by player's actions and decisions. Furthermore, this paper presents the concept of hierarchical quests.

Author Keywords
Quest Generation; Games; Planning; Non-determinism; Interactive Storytelling.

ACM Classification Keywords
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search - *Plan execution, formation, and generation*; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems - *Games*.

INTRODUCTION
Challenges and actions that entertain players are the core of the gameplay provided by the games [3]. Role-playing games (RPGs) in particular deliver challenges through quests, which are a fundamental mechanism for narrative progression and provide players with concrete goals that guide the gameplay. However, most of the current RPGs are still using static quests with plots manually created by game designers. This type of quest reduces the player's sense of agency (*i.e.* the sense of the player as the agent of his/her own actions) if the authors are not able to anticipate all the player's actions during the development of the game. In addition, traditional quests often fail in providing the player with the ability of interfering in the main plotline of the game. Usually, they offer only localized consequences in the narrative without real effects in main storyline.

Recently, the automatic generation of quests has attracted the attention of the research community [7, 26, 36]. However, most of the approaches that have been proposed are designed mainly to offer challenges to players without concerning about the connections these challenges have with the main storyline of the game. Research on interactive storytelling has been exploring the dynamic generation of interactive narratives since the 1970s and may provide the proper foundation for the creation of dynamic quests with focus on the dramatic results of the player's actions.

The most robust interactive storytelling systems rely on artificial intelligence techniques, such as planning [17], to dynamically generate the sequence of narrative events rather than following predefined branching points. The techniques that support the dynamic generation of stories are also useful to maintain the logical coherence of the entire narrative, which can be divided in multiple chapters, where each chapter could be a different quest in a game context. In addition, such techniques also support the propagation of changes introduced by the players, allowing them to effectively interact and change the unfolding stories. However, applying these techniques in a game is not an easy task, quest generation must intercalate planning and execution to satisfy the real-time performance requirements of highly interactive game environments. Quests must be planned incrementally, plans must be monitored, and they must be revised as the game world changes.

In this paper, we explore the use of narrative generation techniques in the game context and present a new method for the generation of dynamic quests based on hierarchical task decomposition and planning under non-determinism. The proposed approach is capable of generating and controlling the execution of simple and complex quests defined in terms of logical operators and multiple authorial goals. The method also supports quests with multiple endings, which allows players to effectively affect the main

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACE '14, November 11 - 14 2014, Funchal, Portugal
Copyright 2014 ACM 978-1-4503-2945-3/14/11…$15.00
http://dx.doi.org/10.1145/2663806.2663833

storyline of the game while maintaining its flow and coherence. The main objective of this paper is to present this method and to validate its real-time performance on highly interactive game environments.

QUESTS

In literature, quests are the essence of romance. Moreover, multiple adventures can occur in the course of a quest [15, 24]. According to Propp's analysis of folktales [32], the main adventure begins when a villainy is committed or a lack is recognized, after which the hero departs on a perilous journey, culminating with a struggle against some sort of adversary. But other adventures often occur, typically in a preliminary phase wherein the villain makes preparations, and even after the hero's victory the quest may not terminate. Indeed the hero's return, as happens in Homer's *Odyssey*, may be delayed by a series of challenging incidents. And his eventual arrival may involve a confrontation against a false hero, who must be exposed before the true hero receives the prize.

All these adventures are similarly structured tests for the hero and the other characters, as noted by Greimas [18], and the continuation of the story depends on their outcome. And, to terminate successfully, an adventure may require some resource, such as a weapon or a potion, whose obtainment in turn precipitates a secondary adventure, and so on, recursively, as happens in many tales of the *Arabian Nights*.

The quest assumes an even more intricate hierarchical structure if the outcome of a plan to pursue an adventure is non-deterministic. In this paper we examine situations arising from non-determinism, further extended by the possibility, in case of failure, to try alternative actions, leading thereby to a variable adventure scheme.

In a quest game, also named Role-Playing Game (RPG), quests are missions or objectives to be accomplished by avatars (*i.e.* game characters controlled by human players). Basically, there are two RPG types: tabletop RPG and computer RPG. In a tabletop RPG, also known as a pen-and-paper RPG, players assume the roles of characters, and a special person, called the game master (GM), acts as an organizer, arbitrator, and moderator – all of them gathered around a table with cards and dice. The most famous RPG is Dungeons & Dragons, started in 1974 and still being published in several new editions. Computer RPGs started with adventure games in text format in 1975 (originally called MUDs – Multi-User Dungeons). They have evolved to produce top-rated single-player RPGs (*e.g. Skyrim* series [35] and *Mass Effect* series [27]) and multi-player RPGs (*e.g. World of Warcraft* [40]). Modern RPGs are starting introducing some interactive storytelling techniques, notably, *Apocalypse World* [5] as tabletop RPG, and *Mass Effect* [27] as computer RPG. However, the narrative structures of the RPGs are still very simple.

During the last fourteen years, researchers have been intensifying the search for a quest theory in computer games [1, 2, 22, 34, 38, 39]. However, no work has yet fulfilled this challenge, because the concept of narrative in videogames is not properly understood. There are many contradictory opinions about the videogame's narrative: Aarseth [2] describes it as a "post-narrative discourse"; Tosca [38] claims that games are not narratives (and only after the quest has been completed, it can be narrated as a story); and Jenkins [22] defends a hybrid concept between games and narratology. Moreover, researchers point out that computer RPGs lack the richness of the common creation of a story found in tabletop RPGs [38].

The authors of the present paper believe that a promising approach to conciliate narratives with games and interactive storytelling is to provide semantics to narrative events conceptually similar to Proppian functions [32] through *planning models* [10]. What first called our attention was Propp's insistence that his predefined 31 functions, identified as the "constant elements in a tale", constitute a *sequence*, and that "freedom within this sequence is restricted by very narrow limits which can be easily formulated". We realized that, by defining Propp's functions on the basis of the STRIPS method [13], this partial ordering could indeed be easily explained: the *pre-conditions* to perform a certain function would be first fulfilled by the *post-conditions* (effects: i.e. facts added or deleted to/from the current state of the world) of other functions. In turn, narrative plots could then be viewed as the result of *backward-chaining* planners. Specifically we claim that a powerful approach to narrative elements in interactive quests is to develop dynamic nondeterministic planning models. This approach empowers computer RPGs with rich ways of building unexpected stories full of interesting and meaningful choices for the players. The present paper proposes a model to support this approach.

RELATED WORKS

There are several works on quest generation and interactive narratives in the literature. A generic framework for quest generation is presented by Sullivan et al. [36]. In their system, a game manager uses the player history and the current state of the world to dynamically generate and alter the structure of quests. Their rule-based system is based on a library of quests and is able to dynamically recombine them upon generation. During this process, the system filters possible quest entities through preconditions based on player history and current world state to create personalized experiences. Following a similar approach, Lee and Cho [25] propose a quest generation engine using Petri net planning [41], where the quest plot consists of events expressed with Petri net modules, which are selected according to goal and initial states. Also in the game context, Arinbjarnar [6] uses Bayesian networks to create plots for murder mystery games based on a probability map of typical murder mystery novels.

The combination of quest generation and procedurally generated game worlds is explored by Ashmore and Nitsche [7]. Those authors present a quest generator system based on key and lock puzzles that creates quests inside a player-driven procedurally generated 3D world. The quest generation process occurs during the generation of the game world, where possible locations for keys and locks are scored by evaluators that are dependent on the procedurally generated world. In the context of massively multiplayer online role-playing games (MMORPGs), Pita et al. [31] explore the problem of assigning compelling and coherent quests to players. Their approach is based on three player-centric features used to produce relevant quest paths: the player past experiences (quests memories), his relationship to the character assigning the quest, and player attributes, such as performable actions, skills and proximity to the quest information. These features dictate what the player is capable of accomplishing and his past memories are used as a constraint to keep quests within a relevant path.

Some research works deal with quest generation as an offline problem. Li and Riedl [26] present an offline algorithm for adapting human-authored game plotlines according to player's preferences. Their approach uses a plan refinement technique based on partial-order planning to optimize the global structure of the plotline according to input from a player model, maintain plotline coherence, and facilitate authorial intent. Another offline approach is presented by Doran and Parberry [12]. They analyzed over 750 quests from popular RPGs and created a classification of common quests based on their structure. Using this classification, those authors propose a system that generates quests based on the motivation of non-player characters and a randomly chosen strategy.

The use of planning techniques has also been explored in other game genres. A classic example is the Goal-Oriented Action Planning (GOAP) approach [30], which is used to create the intelligence for non-player characters. In a GOAP architecture, characters are defined by a set of goals mapped to a set of trigger conditions. When a goal is triggered by its activation criteria, the system generates a plan to achieve the activated goal. Following a similar approach, Hoang et al. [19] applies Hierarchical Task Network planning (HTN) to encode strategies to coordinate teams of bots in first-person shooter games. The use of HTN combined with the ability of replanning is also explored by Cavazza et al. [9] to control the behavior of virtual agents in interactive storytelling environments.

Previous works focus mainly in the generation of independent quests, without concerning about the way these quests are linked to each other or to the main storyline of the game. In addition, most of the previous methods do not support real player interventions in the game's narrative, only localized interferences that may change the order quest are offered to players. In this paper, we combine hierarchical quest decomposition and planning under non-determinism to efficiently create nondeterministic quests and support interactive and dynamic story plots.

QUEST GENERATION

Our method is based on the definition that a quest consists of set of tasks to be accomplished by the player (e.g. gathering and delivering items, killing enemies, protecting characters). A combination of several story-related quests can be used to create complex narratives. We can represent the structure of the game's narrative as a hierarchy of quests, where the entire game can be described as a single quest composed of several sub-quests – which may also have their own sub-quests (Figure 1). A quest can be decomposed into primitive events (shaded rectangles in Figure 1) and/or sub-quests. In Figure 1, the dotted lines indicate the decomposition and the arrows indicate the direction of the events. When the story is completed we have a total order of the events ($Event_1$ to $Event_7$ in the example of Figure 1). We use the term event to indicate a primitive action that was executed by the game controller or imposed by the player.

Quests are logically modeled to have multiple goal states, so they can be completed in different ways depending on the player's actions and decisions. Consequently, the results of sub-quests can influence the progression of their parent quests and dynamically change the entire storyline of the game.

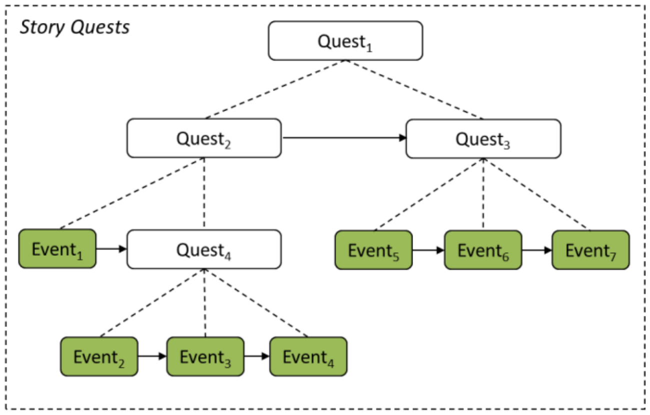

Figure 1. Hierarchy of quests.

Each quest is represented by a planning problem. Therefore, for each active quest in the game, there is an instance of a planner and an instance of a controller, which we call Quest Planner and Quest Monitor respectively.

Figure 2 illustrates the architecture of the proposed quest generator system, which is based on the conceptual model for dynamic planning presented by Ghallab et al. [17]. In our implementation, the Quest Manager is the main module of the system and is responsible for controlling multiple instances of planners and plan monitors. The Quest Planner is responsible for generating a logical plan of actions to achieve an authorial goal of the quest from the current state of the world, while the Quest Monitor is responsible for monitoring the execution of the plan to verify the occurrence of changes introduced by the player. The Game Manager manages the game world by updating the current World State according to the actions performed by the Player during the gameplay, which directly affects the active quests. While performing quests, the Player receives help from the Player's Assistant, who monitors the player progression through the generated quests plans, providing him/her with tips about his/her next objectives and goals. The Quest Library contains a database of quests and sub-quests specified as planning problems, which are dynamically solved by the Quest Planners in real-time.

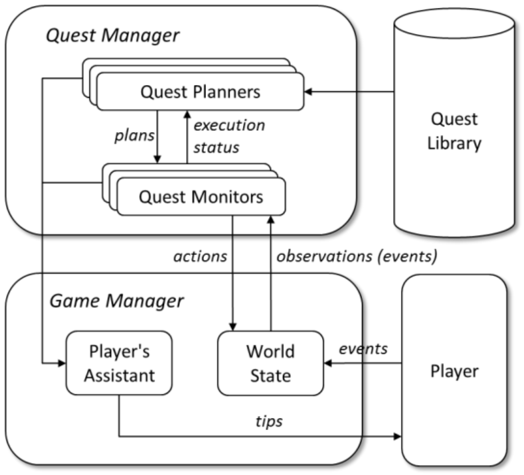

Figure 2. Architecture of the quest generator system.

Hierarchical Quests and the Quest Planner

We define a quest as a planning problem, whose statement is the following tuple:

$$Q = \langle P, O, S_0, H_g \rangle$$

where P is a set of atom symbols (also called *propositions* or *predicates*), O is a set of planning operators, S_0 is the initial state (although our planning system uses the current state of the world as initial state, we keep S_0 in the tuple to be aligned with the formalism of planning processes), and H_g is the hierarchical set of goals, such that:

$S_0 \subseteq P$ is a set of ground literals, where a literal is an atom p or the negation of an atom, $\neg p$, where a negation means to delete the proposition from the current world state S (*i.e.* we use the *close-world assumption*: a proposition that is not explicit specified in a state does not hold in that state);

H_g is a totally ordered set of goals:

$H_g = (\{G_1, G_2, \cdots, G_n\}, \{G_1 \prec G_2, G_2 \prec G_3, \cdots, G_{n-1} \prec G_n\})$, where each goal $G_i \subseteq P$ is a set of ground literals and the order $G_i \prec Gj$ defines the sequence of alternative goals.

An action a is any ground instance of a planning operator $o \in O$, which is the following tuple:

$$o = (name(o), precond(o), effect(o), subq(o))$$

where

- $name(o)$ is an expression of the form $name(x_1, \cdots x_k)$, x_i is a variable symbol that occurs anywhere in o;
- An action a is *applicable* to the current world state S if the preconditions of a (*i.e.* a set of ground propositions) hold in S.
- An action a is *relevant* for a goal G (*i.e.* a can produce a state that satisfies G) if
  (i) the effects of a hold in G, and
  (ii) the effects of a hold in any goal of the sub-quest $q_i \in subq(a)$.

When $subq(o)$ is not empty, o is referred as a *compound operator* otherwise it is a *primitive operator*. An instance of a compound operator is a total-order plan (*i.e.* a totally ordered sequence of actions), which comes from the concatenation of the resolutions of all sub-quests. Each sub-quest is as a classical planning problem.

We consider the cost of doing an action a in a state s as unitary, *i.e.* $\cos t(a,s) = 1$, $s \in S$.

The following examples in a zombie survival game illustrate the hierarchical quests:

```
quest: save-family
s0: character(john), character(anne), place(home),
    place(forest), at(john,forest),
    at(anne,forest), healthy(john),
    infected(anne), safe(home),
    path(forest,home), path(home,forest)
G1: healthy(anne), protected(house)
G2: escaped(john)

Operator: take(CH1, CH2, PL1, PL2)
precond: healthy(CH1), infected(CH2), at(CH1,PL1),
    at(CH2, PL1), path(PL1, PL2), CH1 ≠ CH2
effects: ¬at(CH1, PL1), ¬at(CH2, PL1),
    at(CH1,PL2), at(CH2, PL2)
subq: ∅

Operator: save(CH1, CH2, PL)
precond: healthy(CH1), infected(CH2), at(CH1, PL),
    at(CH2, PL), safe(PL), CH1 ≠ CH2
effects: healthy(CH2), ¬infected(CH2)
subq: save-wife

Operator: protect(CH, PL)
precond: healthy(CH), at(CH, PL)
effects: protected(PL)
subq: protect-house

quest: save-wife
s0: character(john), character(anne),
    character(oldman), place(home),
    place(village), place(hospital),
    place(market), item(antidote1),
    item(antidote2), at(john,home),
    at(anne,home), at(oldman,market),
    healthy(john), infected(anne),
    at(antidote1,hospital),
    has(oldman,antidote2), safe(home),
    path(home,village), path(village,home),
    path(village,hospital),
    path(hospital,village), path(market,village),
    path(village,market)
G1: healthy(anne)
G2: dead(anne)
```

In the above examples we can notice that `john`, `anne` , and `oldman` are characters; `house`, `village`, `market`, `forest`, and `hospital` are places; `john` and `anne` are both at `home`; `john` is healthy, but `anne` is infected; `antidote` is an item that is at the `hospital`; and there is a path connecting `home` with the `hospital` (amongst other paths connecting places). Also we can see that if the compound operator `save` is instantiated as `save(john,anne,home)`, the sub-quest `save-wife` will be triggered, because one of its goal (*i.e.* `healthy(anne)`) is amongst the effects of the action `save(john,anne, home)`.

The game world is logically represented by a state, which consists of a set of ground propositions $S \subseteq P$ defining characters, objects, locations, and their current situation in the game world. If a sub-quest is called, the current state of the world will be used as the initial state of this new sub-quest. Therefore, when the player causes changes in the world, the planner recalculate the quest plan using the modified world state as the initial state of a new classical planning problem. The proposed algorithm can use any classical planner for this step of a simple quest. In the prototype of section 6, we used the HSP2 planner provided by Bonet and Geffner [8] as a STRIPS-like planner.

Any sub-quest is described as an independent planning problem in the Quest Library. The Quest Planner adopts a hierarchy of authorial goals, in the sense that if a higher goal cannot be achieved, the planner tries its successor. The planner can fail to achieve a desired goal either if there is no valid sequence of actions that leads from the initial state to the goal state; or if the prescribed time limit for searching for a solution is exceeded. In both cases, the planner tries to achieve the next successor goal from the authorial goal hierarchy. For example, the hierarchy of goals for the quest `save-family` has two different outcomes that can be described as follows:

```
G1: healthy(anne), protected(house)
G2: escaped(john)
```

where G_1 is the primary goal of the quest that establishes that `anne` must be healthy and the `house` must be protected. If the player modifies the game world in such a way that G_1 becomes unreachable, the planner will try to find a plan to achieve G_2, which requires `john` to save himself escaping from the zombies. In the example above mentioned, if the first goal of the sub-quest `save-wife` fails, then the second goal may be accomplished by the husband killing his wife to save her from the doom of being a walking mindless monster forever. As a general authorial rule, the last successor goal should be always achievable to avoid aborting the story prematurely.

Compound operators represent nondeterministic events that may have different effects on the story plot depending on the player's interferences and decisions while the quest monitor is performing the sub-quests. Although the compound operators may have nondeterministic effects on the quest plan, they are specified with a default list of deterministic effects according to the primary authorial goal of its respective sub-quests. The non-determinism of the compound operators is handled by the Quest Monitor in real-time during the execution of the quest plan.

Once a quest has started, the planning algorithm proceeds by searching in the space of world states for a sequence of actions that leads the player from the current state of the world to one of the quest's goals. However, different from a traditional HTN planning algorithm, and to improve the performance of the planner, our algorithm does not decompose the compound operators during the generation of the initial plan for the quest, what improves the performance of the planner significantly. The planner interprets compound operators as primitive ones, and uses their predefined deterministic effects to generate a plan without instantiating the events of sub-quests. The plans for sub-quests are generated by new instances of quest planners during the execution of the quest plan when the player reaches a compound operator. In this way, our algorithm deals with non-determinism efficiently and gracefully.

Quest Monitor

The Quest Monitor is based on a planning approach that integrates planning, execution, and monitoring. This type of approach can also be found elsewhere [4, 29]. The Quest Monitor works together with the Quest Planner to generate and maintain the coherence of quests in the dynamic and nondeterministic environment of the game.

For each instance of a Quest Planner, there is a Quest Monitor in charge of monitoring the execution of its respective quest plan to verify the occurrence of changes introduced by the player in the game world that violates preconditions of the quest events generated by the planner. In addition, the Quest Monitor is also responsible for instantiating new Quest Planners and Monitors to plan and monitor sub-quests described by the compound operators present in its respective quest plan.

The algorithm for monitoring the execution of quests continuously checks the current state of the world to verify the consistency of the quest plan. If it detects that the current world state is different from the expected state described in the quest plan, it requests a new plan for the Quest Planner using the current state of the world as the initial state for the planning problem. In this way, a new plan to achieve one of the quest goals starting from the current state of the world will be generated. In the new plan, new tasks may be added in order to make the player returns to the previous storyline of the quests or a completely different sequence of events may be created to guide the quests towards a different outcome.

During the execution of the quest plan, the monitoring algorithm also verifies the occurrence of compound operators in the plan. If the next expected event of the quest is defined by a compound operator, a new Quest Planner and a new Quest Monitor are instantiated to handle the process of planning and monitoring the execution of the sub-quest independently. While the player is performing a sub-quest, the Quest Monitor of its parent quest waits until the player has finished the sub-quest to resume the monitoring process.

The process of monitoring the execution of quests and the capacity of replanning the quest's events whenever necessary allows the system to directly support nondeterministic sub-quests with multiple endings that may influence the whole narrative of the game. In nondeterministic sub-quests, player's actions can induce the quest to an outcome that may affect the world state in a way that does not match with the state produced by the predefined deterministic effects of its respective compound operator. Consequently, nondeterministic sub-quests can produce inconsistencies in the plan of their parent quests depending on the way they end. Such inconsistencies will be automatically detected by the Quest Monitor, who will request a new plan to its respective Quest Planner in order to correct inconsistencies and maintain the game flow and coherence. In this way, while performing a sub-quest, the choices made by the player are propagated through the hierarchy of quests, effectively modifying the narrative of the game.

Figure 3 illustrates how player actions can modify the plot of quests and how the combination of planning and monitoring can support nondeterministic quests and handle inconsistencies introduced by the player interventions in the plan of quests. In this example, the player is in a quest to save the life of his family, after his wife was attacked and infected by zombies. *Plan 1* describes the initial plan generated to solve the quest "*Save family*", which consists of taking his wife back home, saving her from the Zombie disease, and protecting his house. The sub-quest "*Save wife*" consists of going to the city hospital, getting the antidote, going back home, and using the antidote to save the life of his wife. Suppose that when the player is trying to go back home with the antidote, he is attacked by a zombie and he breaks the antidote bottle. In this case, the fact `has(player,antidote)` describing that the player has the antidote will be removed from the current state of the world. When this happens, the Quest Monitor of this quest will detect an inconsistency in the quest plan (i.e. the player cannot give the antidote to his wife if he does not have an antidote). In order to solve this inconsistency, a new plan will be requested to the Quest Planner, who finds an alternative plan to achieve the same goal of the previous plan. In *Plan 2*, after breaking the bottle of the first antidote, the player has to go to the market and ask an old man for an antidote, get the antidote, and go back home to save his wife. In this new sequence of events, the task of asking the old man for an antidote involves a dialog where the player has to answer some questions and, depending on the answers, the old man may give or not the antidote. Suppose that the player does not agree with the old man's proposal of giving the antidote in return for the betrayal of all John's friends. In this case, the fact `has(player, antidote)` will not be added to the current state of the world after the player has completed the quest event. In addition, there are no more antidotes available in the game world, what generates a new inconsistency in the quest plan. This inconsistency will trigger another replanning procedure, but now the previous quest goal will not be achievable anymore, which will force the planner to try another authorial goal. In the resulting plan (*Plan 3*), after trying all the alternatives to get an antidote, the only choice the player has is to go home and see his avatar John kill his wife to save her from a dreadful destiny. This new sequence of events affects the resulting world state of the quest "*Save wife*", which introduce an inconsistency in the plan of its parent quest. In this case, the quest "*Protect house*" cannot be executed anymore, because it requires the player's wife to be alive. In order to correct this inconsistency, the Quest Monitor of the parent quest will request a new plan, where the quest "*Protect house*" ends up being replaced by the quest "*Escape*".

PLAYER'S ASSISTANT

An interactive quest with nondeterministic events requires a dynamic mechanism to assist players in their journeys. This assistant may represent an entity in the game that helps the player to complete the game quests, through direct communication acts or through objects and non-player characters. This idea is inspired by the way some games integrate player assistants into the gameplay, like *The Legend of Zelda: Skyward Sword* [37], which includes a spirit called *Fi* that guides the player during his adventure through the game world. *Fi* provides the player with information about locations, enemies, and the tasks the player has to accomplish. This is also explored in the games of the *Metal Gear Solid* series [28], where the player is constantly advised by radio communication about his tasks and objectives. However, none of these games use mechanisms for the dynamic generation of tasks. Usually, player assistants follow predefined scripts.

The Player's Assistant in the proposed architecture is implemented as an agent that is constantly monitoring the player's progression through the quest's plans. In this way, it always knows the next expected events according to the current plan. This information can be directly or indirectly used to help the player to know its next objective in the game. In addition, this information can also be filtered to inform the player only about important facts and tasks.

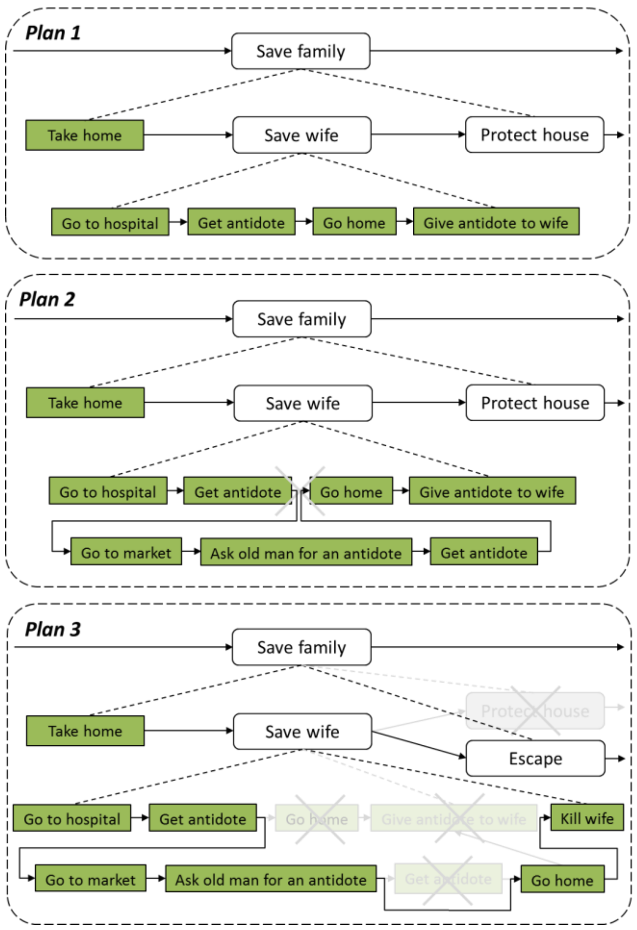

Figure 3. Example of dynamic quest plans generated by the planner while the player is performing actions and progressing through the quest "*Save family*".

The statement of our assistance problem is:

$$H_\tau = \langle E_\tau, T, f \rangle$$

where E_τ is the set of the next expected events at the current time τ, T is the set of all *primitive templates*, and f is a function that filters the set E_τ and returns a more adequate set of events to be announced. The solution of this problem is a set of sentences in natural language that are appropriate for the current time τ.

The events described in the quest plans are represented by logical ground propositions, which must be translated by the Player's Assistant to natural language or other form of representation so they can be easily understood by the player. The process of generating natural language sentences from logical propositions is a well-known task in the area of natural language processing [33]. Although the player assistant could be implemented using any state-of-the-art method of natural language generation [16, 23], we adopt a more straightforward approach based on simple templates. We adopt this approach not only to facilitate the validation of our prototype but mainly because we need real time performance in quest games. Furthermore, recent research has showed that template-based approaches to natural language generation are not necessarily inferior to other approaches [11].

The Player's Assistant uses a set of predefined primitive templates T to generate natural language sentences based on the logical propositions of the plan. For example, the primitive operator `go`, which indicates that a characters must go from one place to another, is associated with the following template:

```
{go}: {CH₁}, you must go to the {PL₂}!
```

where CH_1 and PL_2 are variables that will be replaced by their corresponding parameters of the instantiated primitive operator. For the event `go(john, home, hospital)`, the following sentence will be generated:

```
"John, you must go to the Hospital!"
```

In some cases, variations of the templates depending on the parameters are also necessary to guarantee the correct generation of sentences. In the previous example, if PL_2 is "home", the generated sentence would be "`John, you must go to the Home!`", which would be grammatically incorrect. Such minor inconsistencies are corrected by creating variations of the generic templates with predefined parameters.

In addition, templates can also be combined to create more complex sentences depending on the sequence of quest events. For example, when the player has to go somewhere to get an artifact, it would be more compelling if the player assistant tells the player about the reason he has to go to that place. Situations like that are handled by a *chains of events*, which are represented by *compound templates*. For example, the chain `{go ≺ get}` is represented by the compound template `{{go},{get}}`, which indicates that when a `go` event is followed by a `get`, the sentences generated to describe both events can be joined and presented as a single sentence omitting common elements. In this case, if both actions are performed by the same character, the noun will be omitted in the second sentence. For the sequence of events `{go(john, home, hospital) ≺ get(john, antidote, hospital)}`, the following sentence will be generated:

```
"John, you must go to the Hospital! You need to get an Antidote!"
```

Although, the automated assistant can be able to tell the player about all the actions he must perform to complete quests, this excessive amount of help may compromise the gameplay by providing the player with the solutions of all challenges. This problem can be avoided by filtering the quest's events with the filter f and providing the player only with information about important goals to complete the quest. For example, telling him to get an antidote without indicating where the antidote is. The filter f can be applied to all quests or specific filters may be associated with each quest for a more precise filtering.

The level of help provided by the Player's Assistant can be predetermined by the game designer or automatically adjusted by the game according to the player skills or preferences. In addition, the way the help is presented can also be adjusted to the game style or player preferences. It can be presented at certain points of the game or only when the player request assistance. The information can be presented as dialogs with other characters or as a simple list of tasks to be completed. However, it is important to notice that these tasks are not strict goals. The player is free to explore all possible solutions to complete the quests.

APPLICATION, EVALUATION, AND RESULTS

In order to validate the proposed methods for the dynamic generation of quests, we developed a prototype 2D RPG using the proposed architecture to dynamically generate and control the entire narrative of the game. The narrative pertains to a zombie survival genre and tells the story of a family that lives in a world dominated by a zombie plague. The player controls the brave husband John through several nondeterministic quests to protect his family and save his own life.

The prototype RPG is composed of a total of 26 quests (8 deterministic and 18 nondeterministic) with different hierarchical levels and complexities. The proposed quest generation method combined with this set of quests is capable of generating a considerable number of diversified narratives. In more conventional sequence of events, John's wife is attacked by a zombie and she is saved by John, who finds an antidote. Then, in order to protect his family, John tries to improve the protection of his house, but his daughter end up being attacked by another zombie. After failing in protecting his house, John and his family escape to a remote island, where they have to build a house and find supplies to survive. Unfortunately, some zombies also find their way to the island and attack John and his family again. John survives the attack, but the future of his family is still uncertain. Several different stories with happy, sad, and even dark outcomes can emerge from this basic storyline depending on the player attitudes, decisions, and actions while performing nondeterministic quests. John's wife and his daughter may survive or not, after being infected by a zombie, depending on whether the player succeeds in getting an antidote. After escaping to the island, the player may fail in the quest of finding supplies and one or more members of his family may starve to death. John can even be unable to escape to the island if the player fails in a quest to get fuel to his boat. This unfortunate event will force him to escape to a remote mountain, where a complete different story takes place. Figure 4 shows the sequence of actions the player has to perform in the prototype RPG to complete the quest "*Save wife*", whose plans were previously illustrated in Figure 3.

In the prototype, we used the PDDL language to describe the planning problem [13]. The following example shows the specification of the operator `use-antidote`:

```
(:action use-antidote
  :parameters (?c - character ?x - item
               ?y - character ?w - location)
  :precondition (and (character-at ?c ?w)
                 (healthy ?c)(character-at ?y ?w)
                 (has ?c ?x)(antidote ?x))
  :effect (and (healthy ?y)(not (has ?c ?x)))
)
```

One of the main drawbacks of classical planning algorithms is related to their high computational complexity that grows according to the number of objects and operators involved in the planning problems. In order to evaluate the scalability of the proposed quest generation method, we conducted a performance test comparing the time necessary to generate a valid plan of actions for five quests selected from our prototype RPG with increasing levels of complexity. Table 1 shows some statistics of the selected quests, where the solution cost is given by the number of events of the generated quest ($cost(a,s)=1$).

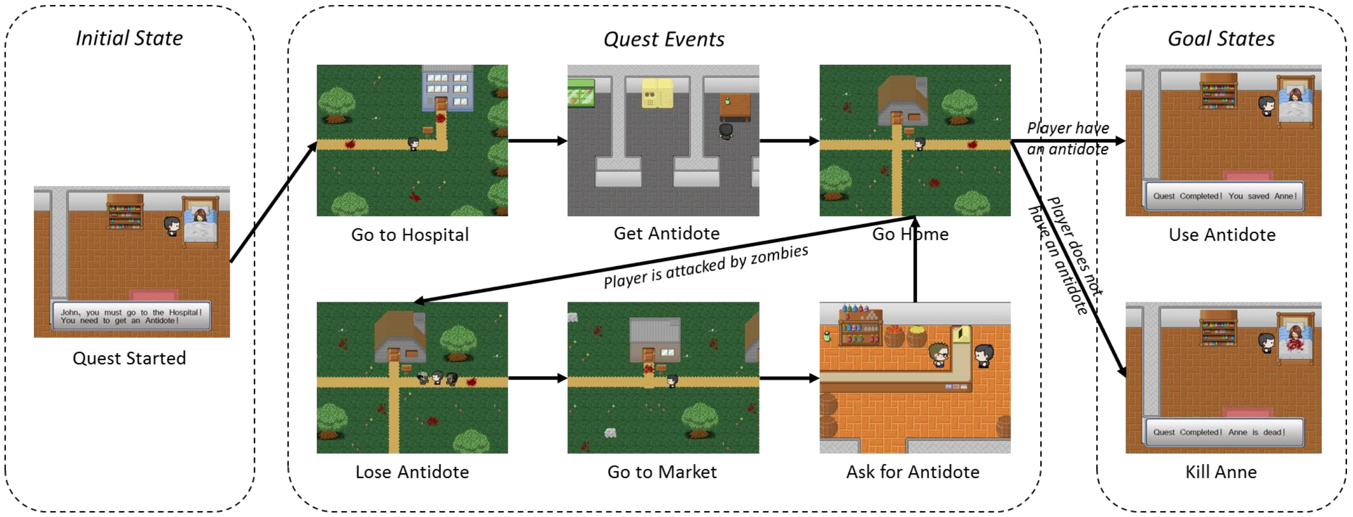
Initial State
Quest Events
Goal States
John, you must go to the Hospital! You need to get an Antidote!
Quest Started
Go to Hospital
Get Antidote
Go Home
Player have an antidote
Player does not have an antidote
Player is attacked by zombies
Quest Completed! You saved Anne!
Use Antidote
Quest Completed! Anne is dead!
Kill Anne
Lose Antidote
Go to Market
Ask for Antidote

Figure 4. Sequence of actions to complete the quest "*Save wife*" in the prototype RPG (see Figure 3).

	Quest 1	Quest 2	Quest 3	Quest 4	Quest 5
Objects	12	16	20	22	36
Operators	4	6	8	9	15
Solution Cost	9	10	17	25	32

Table 1. Evaluated quests with increasing number of objects (characters, locations and items), operators, and solution cost.

For the evaluation test, we created two versions of the selected quests: the first one was defined as a single planning problem, and the second used the proposed approach of dividing quests into sub-quests and then creating plans incrementally as the player progresses through the quests. For the two versions of each quest, we used our planner to find a valid plan of actions to complete the quest according to its initial and goal states. The time spent by the algorithm to complete the task was then calculated and compared. The computer used to run the experiment was an Intel Xeon E5620, 2.40 GHZ CPU, 24 GB of RAM using a single core to process the planning algorithm.

The results of the performance tests are shown in Figure 5. The left bar of each quest correspond to the time spent by the algorithm to find a valid solution to the quest defined as a single planning problem and the right bar indicates the time required to solve the same quest problem defined as a hierarchy of sub-quests. The different colors of the right bars (indicated by numbers) correspond to the time spent by the planning algorithm to solve each sub-quest individually. The performance gain is clearly shown in Figure 5 (it is important to notice that the time is presented in a logarithmic scale).

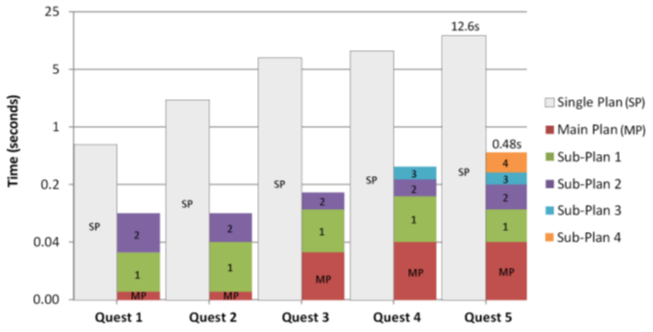

Figure 5. Results of the performance test. The left bars indicate the time spent by the planner to solve the quests as a single planning problem and the right bars represent the total time required to solve them as a hierarchy of sub-quests. Time is presented in a logarithmic scale.

Although our method for dynamically generating quest plans requires replanning procedures every time the player performs an action that modifies the world state in an unexpected way, the efficiency of the quest planner allows the system to update the quest plans in real-time. The player performs the quests without noticeable delays.

The results of the performance experiment confirm the received wisdom that the process of generating quest plans becomes more complex as more objects and operators are added to their respective planning problems. The results also show that generating plans for complex quests using a traditional planning approach quickly become intractable (even for very fast traditional planning algorithms as the HSP2 available at [21]). In our experiment, the process to find a plan to solve Quest 5 spent more than 12 seconds, which is a huge amount of time for a game that must be executed in real-time. Our approach of dividing quests in sub-quests can significantly reduce the amount of processing time required to generate solutions for the quest problems. Using our method, the complete plan to solve Quest 5 was generated in 0.48 seconds. In addition, our approach generates plans incrementally as the player progress through the game, intercalating planning and execution, which divides the planning process in multiple stages.

CONCLUSION

In this paper we presented a new method for the generation of dynamic quests based on hierarchical task decomposition and planning under non-determinism. We propose the concept of hierarchical quests as the basis for that method. The proposed method combines planning, execution, and monitoring to efficiently handle nondeterministic events and support quests with multiple endings that affect the game's narrative. Furthermore, the method creates interactive and dynamic story plots that are directly or indirectly affected by player's actions and decisions at real-time. Also we propose a pragmatic model for assisting the player during his/her quest experience, which is a template-based model adapted to the dynamic quest world.

Our approach provides game designers with new ways of imagining and creating narratives for games using dynamic and nondeterministic quests. We believe that this form of interactive narratives can expand the boundaries of traditional games towards new forms of interactive storytelling, allowing players to make their own decisions and create their own narrative experiences.

Although the developed prototype RPG is still very simple, it demonstrates the capacity of the proposed method to generate and control the execution of nondeterministic quests for games with dynamic narrative plots. However, more tests in more complex scenarios are still necessary. Another import future work is to evaluate some key aspects of the authoring process, including the authoring expressiveness, complexity and the level of control the human author may have over the game's narrative. Furthermore, user studies are also necessary to evaluate our method from the player's perspective.

REFERENCES

1. Aarseth, E. From Hunt the Wumpus to EverQuest: introduction to quest theory. In *Proc. ICEC 2005*, Springer (2005), 496-506.
2. Aarseth, E. Quest Games as Post-Narrative Discourse. In Marie-Laure Ryan (ed.): *Narrative Across Media*. University of Nebraska Press (2004). 361-76.
3. Adams, E. *Fundamentals of Game Design*, New Riders Press, New Jersey, USA, 2013.
4. Ambros-Ingerson, J. and Steel, S. Integrating planning, execution and monitoring. In *Proc. AAAI-1988*, AAAI Press (1988).
5. Apocalypse World, Vincent Baker, Lumpley Games, 2010.
6. Arinbjarnar, M. Dynamic plot generation engine. In *Proc. of the Workshop on Integrating Technologies for Interactive Stories*, ACM Press (2008).
7. Ashmore, C., and Nitsche, M. The Quest in a Generated World. In *Proc. DIGRA 2007*, 503-509.
8. Bonet, B., and Geffner, H. Planning as Heuristic Search. *Artificial Intelligence Special Issue on Heuristic Search*, 129, 1 (2001), 5-33.
9. Cavazza, M., Charles, F., and Mead, S.J. Agents' Interaction in Virtual Storytelling. In *Proc. of the Inter. W. on Intelligent Virtual Agents*, 156-170, Springer (2001).
10. Ciarlini, A.E.M., Pozzer, C.T., Furtado, A.L., Feijó, B. A Logic-Based Tool for Interactive Generation and Dramatization of Stories. In *Proc. ACE 2005*, ACM Press (2005), 133-140.
11. Deemter, K.V., Krahmer, E., and Theune, M. Real versus Template-Based Natural Language Generation: A False Opposition?. *Computational Linguistics*,31,1(2005), 15-24.
12. Doran, J., and Parberry, I. A prototype quest generator based on a structural analysis of quests from four MMORPGs. In *Proc. Inter. Workshop on Procedural Content Generation in Games*, ACM Press (2011), 1-8.
13. Fikes, R. E. and Nilsson, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proc. Inter. Conf. on Artificial intelligence*, 608-620, 1971.
14. Fox, M., and Long, D. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20, 1 (2003), 61-124.
15. Frye, N. *Anatomy of Criticism*. Princeton University Press, Princeton, USA, 1957.
16. Garoufi, K. Planning-Based Models of Natural Language Generation, *Language and Linguistics Compass*, 8, 1 (2014), 1-10.
17. Ghallab, M., Nau, D., and Traverso, P. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers, San Francisco, USA, 2004.
18. Greimas, A. J. *Sémantique Structurale*. Larousse, 1966.
19. Hoang, H., Lee-Urban, S., and Munoz-Avila, H. Hierarchical plan representations for encoding strategic game AI. In *Proc. AIIDE 2005*, AAAI Press (2005).
20. Howard, J. *Quests: Design, Theory, and History in Games and Narratives*. A K Peters Press, Wellesley, USA, 2008.
21. HSP-Planners, The HSP family of planners. Available at: https://code.google.com/p/hsp-planners/
22. Jenkins, H. Henry Jenkins responds in turn: riposte to Game Design as Narrative Architecture. *Electronic Book Review*, 2004.
23. Koller, A. and Petrick, R. Experiences with planning for natural language generation. *Computational Intelligence*, 27, 1 (2011), 23-40.
24. Lacy, N. J. (ed.). *The Grail, the Quest and the World of Arthur*. D. S, Brewer Press, UK, 2008.
25. Lee, Y-S., and Cho, S-B. Dynamic quest plot generation using Petri net planning. In *Proc. WASA 2012 workshop at SIGGRAPH Asia*, 2012, 47-52.
26. Li, B., and Riedl, M.O. An offline planning approach to game plotline adaptation. In *Proc. AIIDE 2010*, AAAI Press (2010).
27. Mass Effect, Mass Effect Series, BioWare, 2012.
28. Metal Gear Solid, Metal Gear Solid Series, Konami, 1998.
29. Myers, K. Towards a framework for continuous planning and execution. In *Proc. AAAI Fall Symposium on Distributed Continual Planning*, 1998.
30. Orkin, J. Three States and a Plan: The AI of F.E.A.R. In *Proc. GDC 2006*, 2006.
31. Pita J., Magerko, B., and Brodie, S. True story: Dynamically generated, contextually linked quests in persistent systems. In *Proc. Conf. Future Play*, 2007, 145-151.
32. Propp, V. *Morphology of the Folktale*. University of Texas Press, Austin, USA, 1968.
33. Reiter, E. and Dale, R. *Building Natural Language Generation Systems*. Cambridge University Press, Cambridge, England, 2000.
34. Rettberg, J.W. Quests in World of Warcraft: Deferral and Repetition. *Digital Culture, Play, and Identity: a World of Warcraft Reader*, MIT Press (2008), 167-184.
35. Skyrim, The Elder Scrolls V: Skyrim, Bethesda Game Studios, 2011.
36. Sullivan, A., Mateas, M., and Wardrip-Fruin, N., Rules of engagement: Moving beyond combat-based quests. In *Proc. Intelligent Narrative Technology Work.*, 2010, 1-8.
37. The Legend of Zelda: Skyward Sword, Nintendo, 2011.
38. Tosca, S. The quest problem in computer games. In: *Proc. TIDSE 2003.* Fraunhofer IRB Verlag Press (2003).
39. Tronstad, R. *Interpretation, Performance, Play, and Seduction: textual adventures in Tubmud*. PhD Thesis, Faculty of Arts Dept. of Media and Communication, Univ. of Oslo, 2004.
40. World of Warcraft, Blizzard Entertainment, 2005.
41. Ziparo, V.A., and Iocchi, L. Petri Net Plans. In *Proc. MOCA 2006*, 267-290.