Player Behavior Modeling for Interactive Storytelling in Games

Edirlei Soares de Lima [1] [*] Bruno Feijó [2] Antonio L. Furtado [2]

Rio de Janeiro State University (UERJ), Department of Computational Modeling, Brazil [1]

Pontifical Catholic University of Rio de Janeiro (PUC-RIO), Department of Informatics, Brazil [2]

Abstract

Video games add new dimensions to traditional storytelling by allowing players to change narratives through their own actions. In modern Role-Playing Games (RPGs), this is usually done by adopting branching storylines based on key choices presented to players at certain points of the game. However, such branching points are usually presented through specific dialog choices or predetermined actions, which reduces the player's sense of agency. Currently, both game industry and game consumers have great interest in new forms of interactive storytelling that may provide games with truly interactive stories, in which all in-game player's actions and behaviors can affect the development of the narrative. In this paper, we explore the combination of player modeling and narrative generation techniques. We propose a novel approach for interactive storytelling in games based on player behavior modeling, hierarchical task decomposition and nondeterministic planning. The proposed method is capable of generating dynamic and nondeterministic quests that are directly or indirectly affected by in-game player behavior, which is modeled in terms of the Big Five factors. The main objective of this paper is to present this method and to validate its precision and real-time performance on highly interactive game environments.

Keywords: Player Modeling, Player Behavior, Quest Generation, Games, Interactive Storytelling.

1 Introduction

Video games add new dimensions to traditional storytelling by allowing players to change narratives through their own actions. In modern Role-Playing Games (RPGs), this is usually done by adopting branching storylines based on key choices presented to players at certain points of the game. Some RPGs, such as *Mass Effect 2* (BioWare, 2010), *Dragon Age: Inquisition* (BioWare, 2014), and *The Witcher 3: Wild Hunt* (CD Projekt RED, 2015) perform this so well that they are able to provide the player a real sense of control over the story. However, such branching points are usually presented through specific dialog choices or predetermined actions (e.g. killing or forgiving an enemy, collecting or not collecting a specific item), which reduces the player's sense of agency (i.e. the sense of the player as the agent of his/her own actions).

Currently, both game industry and game consumers have great interest in new forms of interactive storytelling that may provide games with truly interactive stories, in which all in-game player's actions can affect the development of the narrative. However, interpreting and understanding in-game player behavior in real-time is not an easy task. This problem involves an active topic of research on artificial intelligence, known as player modeling.

Player modeling is the study and use of artificial intelligence techniques [36] for the construction of computational models of players, which includes cognitive, affective and behavioral characteristics. In general, a player model is an abstract description of a player in a game environment [2]. Specifically for the context of behavioral modeling, a player model includes the description of the player's behavior in the game environment. Indeed, the construction of effective player models involves a multidisciplinary intersection of the fields of affective computing, experimental psychology, human-computer interaction, big data, and analytics, which are part of the so called "game analytics" [27].

In recent years, research on player modeling has attracted a lot of attention from both industry and academic research [36][2]. Nowadays, analyzing game data is a common practice widely employed in the game industry to validate level design or improve the player experience [33][27]. Many commercial games are known for adopting player modeling techniques, such as *Silent Hill: Shattered Memories* (Konami, 2009), which dynamically creates personality models of the players and uses them to adapt gameplay elements [23]; *League of Legends* (Riot Games, 2009), which explores the analysis of gameplay data to design new content updates [16]; and *Left 4 Dead* (Valve, 2008), which uses player modeling to adapt the difficulty of the game's challenges in response to the player's actions. Even though player modeling has been successfully applied to commercial games and widely explored by academic researchers, only few works treat the prediction of actual player behavior.

We consider actual player behavior to be the way in which the player acts or conducts him or herself in the game. For instance, the player may behave aggressively, impulsively, or cautiously when facing dangerous situations. Behavior, in general, is not only complex, but also dynamic. A behavioral description for one occasion is likely to be invalid for another occasion. In fact, behavior is susceptible to variation in consequence of any change in time, place, emotion, and social context [20]. In addition, a player's behavior within a game environment may be very different from the same person's behavior when dealing with real world situations.

In this paper, we explore the combination of player modeling and narrative generation techniques. We propose a novel approach for interactive storytelling in games based on player behavior modeling, hierarchical task decomposition and nondeterministic planning[1]. The proposed method is capable of generating dynamic and nondeterministic quests that are directly or indirectly affected by in-game player behavior, which is modeled in terms of the Big Five factors [11]. The main objective of this paper is to present this method and to validate its precision and real-time performance in highly interactive game environments.

The paper is organized as follows. Section 2 reviews related work. Section 3 gives an overview of the system architecture and introduces the testbed game used to validate our method. Section 4 presents the proposed player behavior model. Section 5 describes the proposed method for quest generation based on nondeterministic planning and player behavior modeling. Section 6 contains an evaluation of our method. Section 7 offers concluding remarks.

*e-mail: edirlei.lima@uerj.br

[1] We use the term "nondeterministic planning" for planning problems in which the planning domain is a nondeterministic state-transition system, i.e. an action may have more than one possible outcome [10][14].

2 Related Work

There are several works on player modeling in the literature. One of the earliest attempts to create player models came in 1996 when Richard Bartle [4] proposed his four player types (Achievers, Socializers, Explorers, and Killers). Following Bartle's work, Bateman and Boon [5] created another model using the Myers-Briggs personality indicator [22] to categorized players into four classes: Conqueror, Manager, Wanderer and Participant. However, as previously pointed by Tuunanen and Hamari [32], type-based approaches are very limited, because types provide only a superficial information about the player, which can be even more blurred considering that most players cannot be adequately categorized into a single group.

In recent years there have been several successful implementations of player modeling in games, whose applications include the use of player models for adapting player experience, game balancing, personalized content generation, playtesting analysis and game authoring. Missura and Gärtner [21] explore the use of clustering and classification techniques to dynamically adjust the difficulty of a shooter game. Their method uses k-means and support vector machines to classify players into different types based on gameplay data. Weber and Mateas [35] employ a series of classification algorithms for recognizing player strategies in *StarCraft* (Blizzard Entertainment, 1998). Mahlman et al. [19] use several supervised machine learning algorithms, trained with a set of player behavior data extracted from the game *Tomb Raider: Underworld* (Crystal Dynamics, 2008), in order to predict when a player will stop playing the game and, if the player completes the game, how long will it take to do so. Machado et al. [18] and Spronck and den Teuling [29] explore player modeling in the context of the game *Civilization IV* (Firaxis Games, 2005). Machado et al. [op. cit.] create models of virtual agent's preferences using classifiers based on support vector machines, and Spronck and den Teuling [op. cit.] use a sequential minimal optimization (SMO) classifier to build a player model to predict specific preference values. In a recent work, Valls-Vargas et al. [33] propose a player modeling framework to capture and predict play style using episodic segmentation of gameplay traces and sequential machine learning techniques. Their framework utilizes multiple models that include predictions from previous time intervals to identify how players change play style over time.

The use of player modeling has also been explored in interactive storytelling systems. Barber and Kudenko [3] present an interactive story generator system that learns the personality of its users by applying predefined increments or decrements to a vector of personality traits, such as honesty and selfishness, in response to the users' decisions. Seif El-Nasr [26] presents an interactive storytelling system called Mirage, where both player behavior and personality are modeled in order to allow users to participate in a more engaging drama. The system tracks user's actions to adjust a vector of values representing tendencies toward character traits (heroism, violence, self-interestedness, and cowardice). Sharma et al. [28] present an interactive storytelling system that combines past captured game traces and player survey data to create player models, which are used to dynamically determine the next plot point that is best suited to specific users. Thue et al. [31] present PaSSAGE, an interactive storytelling system that uses player modeling to automatically learn a model of the player's preferences through observations of the player in the virtual world, and then uses the model to dynamically select the content of an interactive story. The player is modeled by a vector, where each dimension is the strength of one of the Laws' stereotypes [15]. As the player performs actions, dimensions are increased or decreased in accordance to predefined rules. Ramirez and Bulitko [25] use this player model with a reward function in such a way that, when several narratives are generated, the one that maximizes this function is automatically selected.

The Big Five model was used in some previous works on player modeling. Van Lankveld et al. [34] investigate whether a personality profile can be determined by observing the player's behavior in a customized scenario for the game *Neverwinter Nights* (Bioware, 2002). They adopted the Big Five model to define the player's personality profile. In a recent work, Nagle et al. [23] explore the application of the Big Five model for difficulty adjustment in a first-person shooter game. They present a linear regression model to predict difficulty adaptations that maximize enjoyment and gameplay duration based on player personality.

Even though player modeling has been widely explored in games, little work has been done to use the player's behavior to adapt game narratives. In addition, most previous works on behavior modeling are based on very simplified models of player archetypes, which fail in representing blended behaviors, as well as in providing more detailed information about the actual player behavior. The few previous works that explore the Big Five model for player modeling use it only to establish personality profiles.

3 System Overview

The player behavior model proposed in this paper was built as an extension of our previous work on hierarchical quest generation [17]. In our system, the structure of the game's narrative is represented 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.

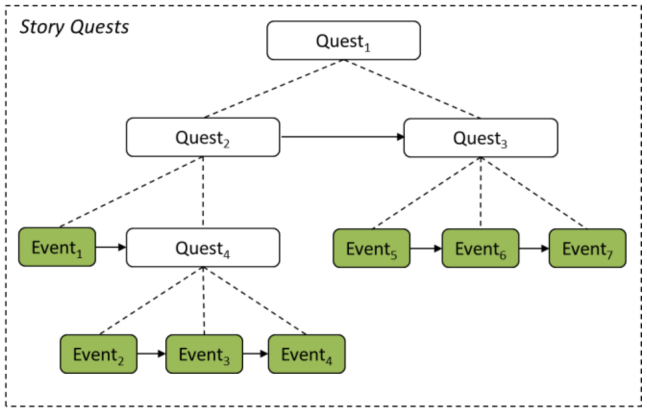


Figure 1: Hierarchy of quests.

Quests are logically modeled to have multiple goal states, so they can be completed in different ways depending on the player's actions. Consequently, the results of sub-quests can influence the progression of their parent quests and dynamically change the entire storyline of the game. This dynamism is achieved by modeling each quest as a nondeterministic planning problem.

3.1 System Architecture

Figure 2 illustrates the architecture of our system, which is consistent with the conceptual model for dynamic planning presented by Ghallab et al. [10]. In our implementation, the Quest

Manager 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 starting from the current state of the world. 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 *events* produced by actions performed by the Player during the gameplay. In addition, the Game Manager maintains the Player Model, which is updated with *input data* extracted from the World State. While the World State aggregates all information about the events that occur in the game as result of direct player actions, the Player Model maintains a more general description of the actual in-game player behavior, including the most recent score of the Big Five factors and the average score accumulated over time. The observations (*events* and *behaviors*) produced by the Player Model and the World State can directly affect the active quests. While performing quests, the Player receives help from the Player's Assistant (described in more detail in [17]), which monitors the player progression through the generated quest 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.

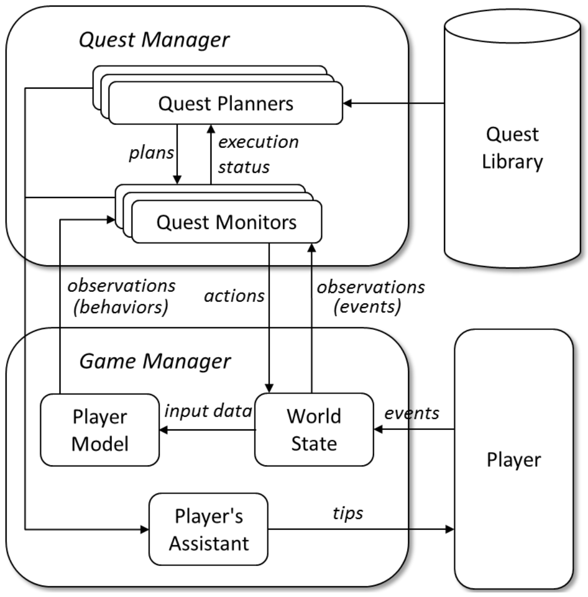


Figure 2: Architecture of the quest generator system.

3.2 Prototype Game

The game used to test and validate our method is a 2D RPG (Figure 3) that uses 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 game is composed of 26 quests (8 deterministic and 18 nondeterministic) with different hierarchical levels and complexities. In the baseline story, John's wife is attacked by a zombie and is saved by John, who finds an antidote. Then, in order to defend his family, John tries to improve the protection of his house, but his daughter ends 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 behaviour and decisions 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 he fails in a quest to get fuel for his boat. This unfortunate event will force him to escape to a remote mountain, where a different story takes place.

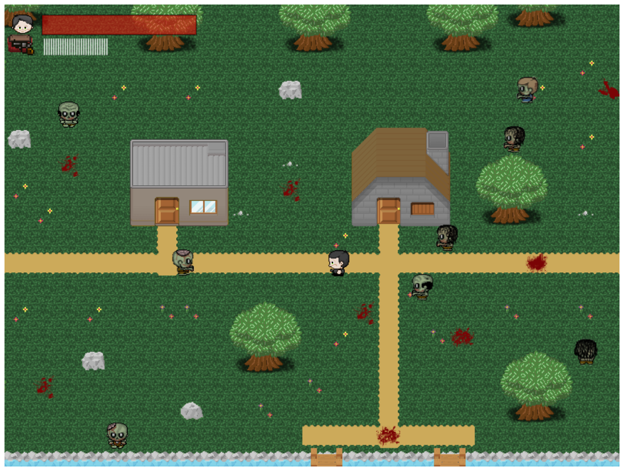


Figure 3: Prototype game used to test and validate our method.

The gameplay of the prototype game is driven by the story quests, wherein the player has to collect items, interact with non-player characters and kill enemies (zombies). In order to fight against the zombies, the player has a gun with a limited amount of ammunition, which is reloaded when the player collects ammunition kits. When the player is attacked by zombies, he/she loses an amount of life (i.e. of the life energy initially attributed to the player), which is only restored when he/she collects medic kits. In addition to the enemies, the player finds through the game two types of non-player characters: (1) normal non-player characters, which are characters that talk and interact with the player; and (2) non-player characters in dangerous situations, which are characters that can be saved by the player.

4 Player Behavior Model

The process to create a computational model capable of recognizing the current player behavior using only gameplay data is a complicated task. Human behavior is the result of complex reflective-impulsive processes [30], which are influenced by a series of factors (e.g.: personality traits, beliefs, and cultural aspects). In addition, behavior is also dynamic, meaning that it is susceptible to vary with time, place, situation, and context [20]. A model built to predict player behavior must be able to handle, not only its complex nature, but also its natural dynamism.

As illustrated in Figure 3, a general player behavior model is composed of three main components: input, output, and a function. The model's input comprises a set of observations extracted from the gameplay data, which should convey enough information about the player's behavior. The model's output represents the set of behaviors that can be predicted by the model based on the input observations. The model's function is the core of the model – it maps the input observations into the output behaviors. The next sub-sections describe how these general components are implemented in the proposed behavior model.

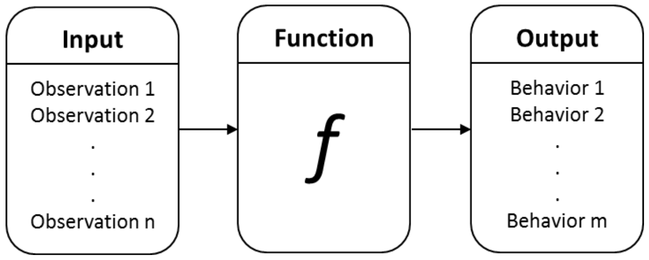


Figure 3: Components of a general player behavior model.

4.1 Input

Since behavior is dynamic and time-dependent, the model must be able to recognize all behavioral changes that occur over time. Consequently, the system must be constantly capturing gameplay data and using it as input to the model. This dynamic process is performed using time windows, which are constant time intervals where the gameplay data is collected and then used to predict the current player behavior. The length of the windows is a crucial variable to determine the precision of the model. Too short time windows may provide only a limited amount of information about the player behavior, but too long windows may fail in capturing transitions between behaviors and produce blurred data. In order to determine the best length for the time windows, we conducted several tests with window sizes of varying length. The results of these experiments are presented in section 6.

The data used as input to a player model is composed of a set of statistical features extracted from the gameplay during a time window. Although these features are dependent of the game mechanics, we selected a collection of general gameplay features that can be found not only in our prototype game but also in other game genres, such as shooters, action-adventures, and role-playing games. In addition, our model of the features is fully adaptive, meaning that the features are computed according to the context from which they were extracted. For example, if a time window comprises the player's actions in a location with 8 enemies and another window includes his actions in a location with 2 enemies, all features related with the number of enemies must take this information into account. If the player kills all enemies in both time windows, the feature that represents the number of enemies killed must be the same for both cases.

The gameplay features used as input to our model are described in Table 1, where T is the length of the time window in seconds, E is the total of enemies seen by the player during the time window, A is the total of ammunition kits seen by the player, M is the total of medic kits seen by the player, N is the total non-player characters seen by the player, and D is the total non-player characters in danger seen by the player.

4.2 Output

The next step to create a player behavior model consists in defining its output, which will represent the possible behaviors that the model will be able to predict in the future. The majority of previous works on player modeling adopt very simplified behavior models based on limited sets of player archetypes [32], which can be very restricted considering that players can exhibit interchangeable and unique blended behaviors. In addition, they often fail in providing more detailed information about the behaviors, such as their intensity.

Table 1: Gameplay features.

ID	Description
F_1	Percentage of time that the player is standing still (in relation with T)
F_2	Percentage of time that the player is walking (in relation with T)
F_3	Percentage of time that the player is colliding (in relation with T);
F_4	Total of new areas explored by the player
F_5	Total of shots fired by the player during the time window
F_6	Percentage of shots that hit targets (in relation with F_5)
F_7	Percentage of shots that miss targets (in relation with F_5);
F_8	Percentage of enemies killed by the player (in relation with E)
F_9	Average time interval between shots fired by the player
F_{10}	Standard deviation of the time intervals between shots fired by the player
F_{11}	Average distance in which enemies were killed by the player
F_{12}	Standard deviation of the distances in which enemies were killed by the player
F_{13}	Average distance in which enemies were hit by shots fired by the player
F_{14}	Standard deviation of the distances in which enemies were hit by shots fired by the player
F_{15}	Average time spent by the player to kill enemies after seeing them
F_{16}	Standard deviation of the times spent by the player to kill enemies after seeing them
F_{17}	Percentage of medic kits collected by the player (in relation with M)
F_{18}	Percentage of life the player recovered without necessity (i.e. by using medic kits when the player's life was almost full) (percentage calculated in relation with F_{17})
F_{19}	Average time spent by the player to collect medic kits after seeing them
F_{20}	Standard deviation of the times spent by the player to collect medic kits after seeing them
F_{21}	Percentage of ammunition kits collected by the player (in relation with A)
F_{22}	Percentage of ammunition kits used by the player without necessity (i.e. by using ammunition kits when the gun clip was almost full) (percentage calculated in relation with F_{21})
F_{23}	Average time spent by the player to collect ammunition kits after seeing them
F_{24}	Standard deviation of the times spent by the player to collect ammunition kits after seeing them
F_{25}	Percentage of non-player characters with whom the player interacted and talked (in relation with N)
F_{26}	Average time spent by the player to talk with non-player characters after seeing them
F_{27}	Standard deviation of the times spent by the player to talk with non-player characters after seeing them
F_{28}	Percentage of non-player characters in danger saved by the player (in relation with D)
F_{29}	Average time spent by the player to save non-player characters in danger after seeing them
F_{30}	Standard deviation of the times spent by the player to save non-player characters in danger after seeing them
F_{31}	Total damage suffered by the player during the time window (i.e. total life loss)
F_{32}	Total of times the player changed his direction during the time window
F_{33}	Average time interval between the moments when the player changed directions
F_{34}	Standard deviation of the time intervals between the moments when the player changed directions

To define a more general and robust output for our behavioral model, we adopted a widely accepted theory about human personality: the *Five Factor Model* (also known as "Big Five") [11]. Big Five is a dimensional representation of human personality structure, which claims that, by using five personality traits, it can suitably account for personality diversity. The Big Five factors are:

(1) **Openness:** those who are high on this factor are imaginative, curious and open to new ideas. In contrast, those who score low on this factor are indifferent and uninterested;

(2) **Conscientiousness:** the ones that display high degree of this factor are meticulous, efficient and systematic. Who scores low is careless, chaotic and disorderly;

(3) **Extraversion:** high scorers are characterized by high indulgence in social activities. On the opposite side, low scorers are reserved and shy.

(4) **Agreeableness:** a high score on this factor characterizes helpful, co-operative and friendly people. In contrast, low score characterizes selfish and hostile people.

(5) **Neuroticism:** those who score high on this factor are emotionally unstable, anxious and aggressive. In contrast, those who score low are well-adjusted and calm.

The five dimensions of the human personality structure are supported by several questionnaires, inventories, and adjective rating scales designed to measure each dimension (e.g.: [9][8][12]). Personality classification is then achieved by assigning five numerical scores (one per dimension) that account for how well each factor describes the person. The attribution of the scores is typically performed with questionnaires that consider observable behavior and characteristics of the individual.

Although the Big Five factors are ordinarily used to describe individual personality, they can also be correlated with specific behaviors. In fact, past studies have pointed that several human behaviors can be adequately explained in terms of the Five Factor Model [24][1]. One example of behavioral taxonomy based on the Big Five is presented by Back et al. [1], who assigned a multitude of concrete actual behaviors to each of the five dimensions of the Big Five on the basis of a systematic investigation of theoretical and empirical approaches to personality and social behavior.

The output of our model is represented by the Big Five factors, which are disposed on five behavioral axes (Figure 4), each within the interval of [-1, 1]. We adopted the taxonomy proposed by Back et al. [1] to define the general behavioral aspects of each factor (Table 2), which we divided into positive (+) and negative (-) behaviors in accordance with the factor's score. The sign (- or +) does not mean destructive or constructive behaviors, but simply indicates the two opposite sides of the Big Five dimensions (i.e. low and high scores). Each behavioral aspect is also associated with a set of general in-game player behaviors, which describes concrete player behaviors within a game environment.

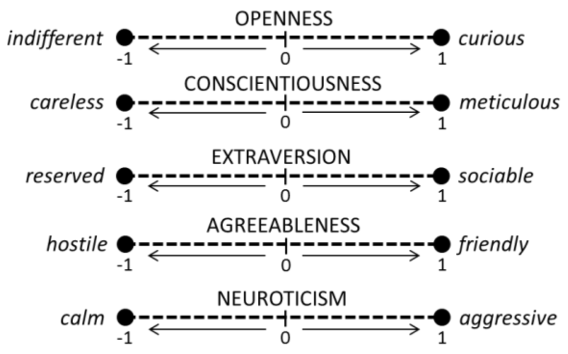


Figure 4: Big Five factors represented as behavioral axes.

4.3 Function

Once we have defined the input and output of our model, we need to establish a function capable of learning and predicting player behaviors. Considering that the output of our model comprises five numerical values representing the Big Five factors, the task to build this function can be seen as a multi-output regression problem [7]. Existing methods to handle this type of problem can be categorized as: (1) problem transformation methods, where the multi-output problem is converted into independent single-output problems, which are solved using a single-output regression algorithm; and (2) algorithm adaptation methods, which adapt single-output methods to directly handle the multi-output data. Among these methods are those using Artificial Neural Networks [13], which work as a typical multi-output regression algorithm to handle problems where the outputs are independent of each other.

Table 2: General behavioral aspects of the Big Five factors.

Big Five Factors	Behavioral Aspects	In-game Player Behavior
Openness	+ curious, interested, inquisitive	• Explores the environment • Collects all the available items
	- indifferent, incurious, uninterested	• Explores only indispensable parts of the environment • Collects only indispensable items
Conscientiousness	+ meticulous, efficient, systematic	• Rarely gets attacked by enemies • Rarely misses a shot • Collects and uses items only when they are needed
	- careless, chaotic, disorderly	• Frequently gets attacked by enemies • Frequently misses shots • Collects and uses items when they are not needed
Extraversion	+ sociable, talkative, active	• Frequently interacts with non-player characters • Interacts with non-player characters as soon as possible
	- reserved, shy, passive	• Rarely interacts with non-player characters • Postpones interactions with non-player characters
Agreeableness	+ friendly, altruistic, helpful	• Always tries to save non-player characters that are in danger
	- selfish, hostile, obstinate	• Rarely tries to save non-player characters that are in danger
Neuroticism	+ aggressive, nervous, unstable	• Tries to kill all enemies • Performs disordered movements
	- calm, relaxed, balanced	• Kills only threatening enemies • Performs only necessary movements

In the proposed system, we implemented the model's function using an Artificial Neural Network trained to predict the values for the Big Five factors based on the statistical features extracted from the gameplay (Table 1). More specifically, we employed a single hidden layer Neural Network trained by an incremental back-propagation learning algorithm using a sigmoidal activation function. In our experiments, we used 64 neurons in the hidden layer. The algorithm was implemented using the FANN library[2].

Since our method employs a supervised machine learning technique to create the player model, samples of gameplay sessions need to be captured and annotated by an expert with labels describing the current player behavior. Considering that our model characterizes the player behavior with five numerical scores, each representing one of the Big Five factors, this annotation process must cover all characteristics of the observable behavior and measure each score systematically. To standardize this process and assist the human expert during the annotation process, we created a simple training questionnaire based on the general in-game player behaviors that contribute to the scores of each Big Five factor in accordance with the rating scales of the Revised NEO Personality Inventory (NEO PI-R) [8]. Our questionnaire is composed of 10 statements regarding the observable player behavior. Each statement is followed by a ten-point Likert scale on which the expert has to rate how much of the behavior indicated by the statement he/she could observe on the analyzed gameplay segment. The scale ranges from "not even a little" (-5) through "neutral" (0) to "a lot" (5). Each statement contributes to the measurement of at least one of the Big Five factors used to characterize the player behavior. The full questionnaire is available in a separate online document[3].

In order to gather samples to train our Neural Network, we recorded the gameplay sessions of 52 players, which generated approximately 4 hours of gameplay. For all sessions, the system automatically captured the statistical features used by our model in five time windows of different lengths (5, 10, 15, 20 and 25 seconds) and recorded a video of the game screen. As previously mentioned, different time windows are being used in order to determine the best length for them.

After capturing the data, each sample (the set of features for a particular time interval) was associated with its respective video segment (i.e. the video segment extracted from the game screen video at the exact time interval during which the features were captured). Then, three voluntary human experts analyzed all video segments and used the training questionnaire to measure and annotate the scores of the Big Five factors for the samples of each time window. Each sample was analyzed by one expert. Five datasets were created (one for each time window). The numbers of samples of the datasets are: (1) time window of 5 seconds – 2903 samples; (2) time window of 10 seconds – 1451 samples; (3) time window of 15 seconds – 967 samples; (4) time window of 20 seconds – 720 samples; and (5) time window of 25 seconds – 562 samples.

After creating and selecting the best dataset, the Neural Network can be trained offline and then used to predict the player behavior in real-time. An evaluation of the precision and performance of the Neural Network is presented in section 6.

5 Quest Generation

Challenges and actions that entertain players are the core of the gameplay of many 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. Even modern RPGs that have quests with multiple outcomes, usually implement them using predefined branching storylines based on key choices. This type of quest reduces the player's sense of agency if the designers are not able to anticipate all the player's actions during the development of the game. In addition, traditional quests often fail to provide to the player the ability to interfere in the main plotline of the game.

5.1 Hierarchical Quests and the Quest Planner

We define a quest as a *planning problem*, expressed by the 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 in conformity to 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, $\neg p$, letting negation signify the deletion of the proposition from the current world state S (*i.e.* we use the *close-world assumption*: a proposition that is not explicitly 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 $o \in O$, denoted by the 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 hold in S.
- An action a is *relevant* for a goal G (*i.e.* a can produce a state that satisfies G) if the effects of a hold in G, and 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), resulting from the concatenation of the resolutions of all sub-quests. Each sub-quest is handled 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: ∅
```

[2] Fast Artificial Neural Network Library - http://leenissen.dk/fann/
[3] http://www.icad.puc-rio.br/~logtell/interactive-quests/quest1.pdf

```
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 one of 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 recalculates 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 our implementation, we used the HSP2 planner provided by Bonet and Geffner [6], which is fully compatible with our STRIPS-like formalism.

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 the intended goal cannot be achieved, the planner tries its immediate successor in the hierarchy. 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. above mentioned example, 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 nondeterministic nature 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, differently 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, which would significantly affect the performance of the planner. The planner interprets compound operators as primitive, and uses their predefined deterministic effects to generate a plan without instantiating the events of sub-quests. When the player reaches a compound operator, the plans for sub-quests are generated by new instances of quest planners. In this way, our algorithm deals with non-determinism efficiently and gracefully.

The player behavior can be used as precondition for both primitive and compound operators. Two sets of five especial propositions are used to represent the players' recent and average scores of the Big Five factors. While the first set of propositions (`openness`, `conscientiousness`, `extraversion`, `agreea-bleness`, and `neuroticism`) refers to the last player behavior observed (predicted by the model), the second set (`avg-openness`, `avg-conscientiousness`, `avg-extraversion`, `avg-agreeableness`, and `avg-neuroticism`) represents the average scores accumulated over time for the current player.

The following examples illustrate the usage of the player behavior as precondition for operators:

```
Operator: give(CH1, CH2, IT, PL)
precond: at(CH1,PL), at(CH2,PL), healthy(CH2),
    has(CH1,IT), avg-agreeableness(CH2,X), X > 0.5
effects: has(CH2,IT), ¬has(CH1,IT)
subq: ∅

Operator: not-give(CH1, CH2, IT, PL)
precond: at(CH1,PL), at(CH2,PL), healthy(CH2),
    has(CH1,IT), avg-agreeableness(CH2,X), X < 0.5,
    avg-neuroticism(CH2,Y), Y < 0.5
effects: ¬has(CH2,IT)
subq: ∅

Operator: kick-out(CH1, CH2, IT, PL)
precond: at(CH1,PL), at(CH2,PL), healthy(CH2),
    has(CH1,IT),avg-agreeableness(CH2,X),X < -0.5,
    avg-neuroticism(CH2,Y), Y > 0.5
effects: ¬has(CH2,IT)
subq: ∅
```

The examples show three different operators that can occur after an `ask` event, where the player (`CH2`) asks another character (`CH1`) for an item (`IT`) in a specific place (`PL`). The operator `give` can only occur if the player has been behaving in a friendly manner towards the others (with an average score of the agreeableness factor higher than 0.5). Otherwise, if the average score of the agreeableness factor be lower than 0.5 and the average score of the neuroticism factor also be lower than 0.5 (meaning that the player is not being very friendly, but is not behaving aggressively), the operator `not-give` can occur. However, if the player has been hostile (agreeableness factor lower than -0.5) and aggressive (neuroticism factor higher than 0.5) with others, then `kick-out` is the only operator applicable.

5.2 Quest Monitor

The Quest Monitor is based on a planning approach that integrates planning, execution, and monitoring. 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. Its job is to verify the occurrence of changes introduced by the player in the game world that violate 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 handle 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 and the player behavior model 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. Similarly, if changes in the current or in the average player behavior are detected, a new plan is requested. In this way, a new plan to achieve one of the quest goals will be generated. In this plan, new tasks may be added in order to make the player return 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 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, with the capacity of replanning the quest's events whenever necessary, allows the system to directly support nondeterministic sub-quests with multiple endings so as to 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 the state produced by the predefined deterministic effects of its respective compound operator. Consequently, nondeterministic sub-quests can introduce inconsistencies in the plan of their parent quests depending on the way they end. Such inconsistencies will be automatically detected by the Quest Monitor, which will request a new plan to its respective Quest Planner, in order to correct inconsistencies and maintain the flow and coherence of the game. In this way, while performing a sub-quest, the choices made by the player are propagated through the hierarchy of quests, effectively modifying the game's narrative.

Figure 5 illustrates how player actions and behaviors can modify the plot of quests, and how the combination of planning and monitoring can support nondeterministic quests and handle inconsistencies introduced by player's 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 the player's 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 wife's life. Suppose that, when the player is trying to go back home with the antidote, he is attacked by a zombie and breaks the antidote bottle. In this case, the fact `has(player, 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 it). In order to solve this inconsistency, a new plan will be requested to the Quest Planner, in an attempt to provide an alternative way to achieve the same goal of the previous plan.

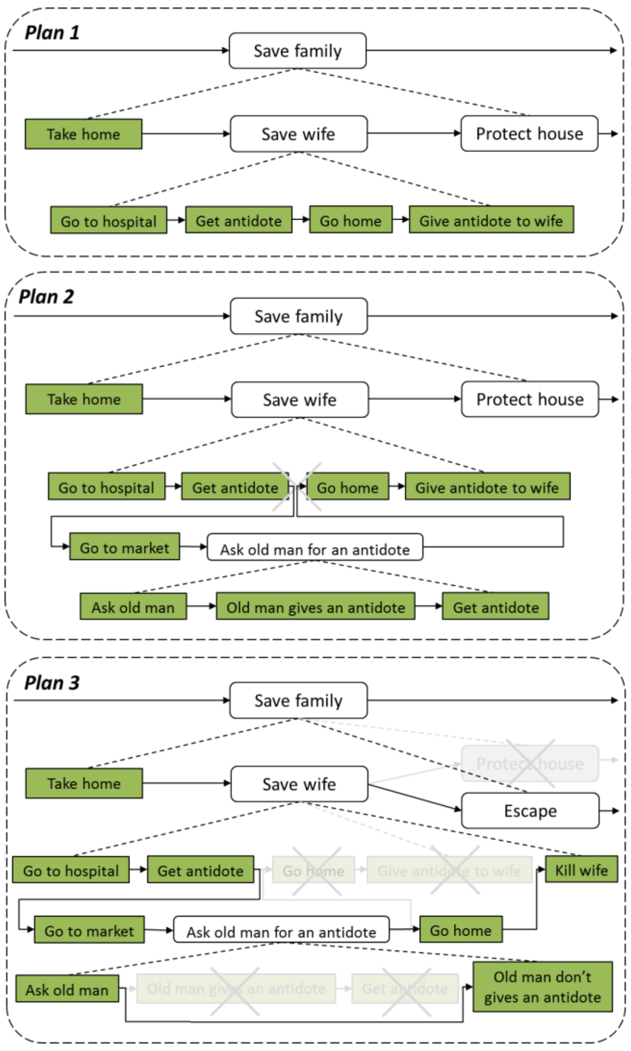


Figure 5: Example of dynamic quest plans generated by the planner while the player is performing actions and progressing through the quest "Save family".

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 another antidote, get the antidote, and go back home to save his wife. However, in this new sequence of events, the event where the old man gives an antidote to the player has a precondition that indicates it can only occur if the player has been behaving in a friendly manner – which should have been true when *Plan 2* was generated. But suppose that, before asking the old man for an antidote, the player starts to behave in a more hostile manner (e.g. by not helping other characters). This behavioral change will cause the player model to be updated and the score of the Big Five factor that represents the agreeableness dimension will be decreased. When this happens, the Quest Monitor of this quest will detect this inconsistency and trigger another replanning procedure. But now the previous quest goal will no longer be achievable, because there are no antidotes available in the game world. This 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 in turn introduces an inconsistency in the plan of its parent quest. In this situation, 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*".

6 Evaluation and Results

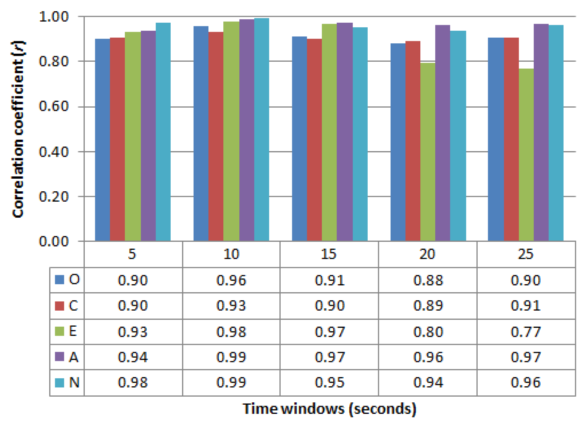
Considering that our general approach for hierarchical quest generation was already evaluated in our previous work [17], we focus here on the evaluation of the player behavior model. For this evaluation, we performed two tests: (1) a precision test to check the accuracy of the proposed model; and (2) a performance test to evaluate the real-time performance of the Neural Network used to predict the player behavior.

For the precision test, we used five datasets of different time windows to train and test our Neural Network. As described in section 4.3, these datasets were created with data collected from 52 gameplay sessions (approximately 4 hours of gameplay) and include samples with all the gameplay features used as input to our model, as well as the scores of the Big Five factors used to characterize the player behavior (model's output). In all the experiments, we used a 10-fold cross-validation strategy.

Three statistical criteria were applied to evaluate the precision of our model: (1) the *root-mean-square error* (*RMSE*), which is the square root of the average squared distances between the actual score and the predicted score (prediction error); (2) the *correlation coefficient* (*r*), which measures the linear association between the actual score and the predicted score; and (3) the *coefficient of determination* (R^2), which represents the proportion of the variance in the actual score that is predictable. The correlation coefficient (*r*) is represented in the interval of [-1, +1]. While an *r* of +1 indicates that the actual score and the predicted score are perfectly related, an *r* of -1 indicates that the two scores are totally unrelated. The coefficient of determination ($R^2 \in [0,1]$) can be thought of as a percentage that indicates the extent to which the scores are predictable. A higher R^2 is an indicator of better fitness for the observations. For the *RMSE* criteria, low values indicate low prediction errors.

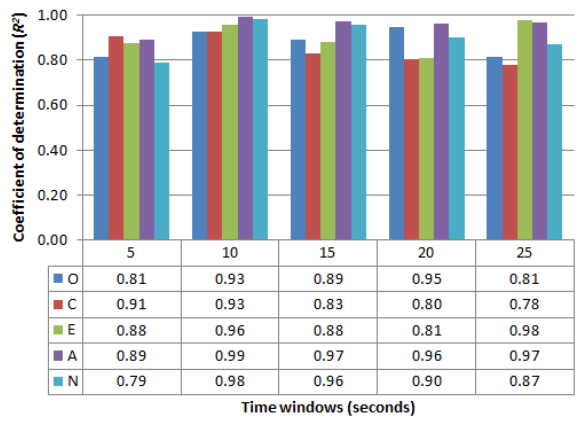
The results of the precision tests are shown in Figures 6, 7 and 8, where each bar represents the average value (10-fold cross-validation) for the evaluation criteria (*r*, R^2, and *RMSE)* calculated for each Big Five factor obtained by Neural Networks trained with datasets of different time windows (5, 10, 15, 20 and 25 seconds). The results indicate that the best length for the time window is 10 seconds (average *r* of 0.97, average R^2 of 0.96, and average *RMSE* of 0.06). The second best length is 15 seconds (average *r* of 0.94, average R^2 of 0.91, and average *RMSE* of 0.08).

To evaluate the performance of our model, we performed the prediction of the player behavior during 5 gameplay sessions, wherein a total of 120 behavior predictions were performed (time window of 10 seconds). For each prediction, we computed the time necessary to calculate the input features and predict the Big Five factors using the Neural Network. The computer used to run the experiment was an Intel Core i7 2630QM, 2.0 GHZ CPU, 16 GB of RAM using a single core to process the Neural Network. As a result, we got an average time of 4.2 milliseconds (standard deviation of 1.2 milliseconds), which indicates the applicability of the proposed method in highly interactive game environments without noticeable delays.



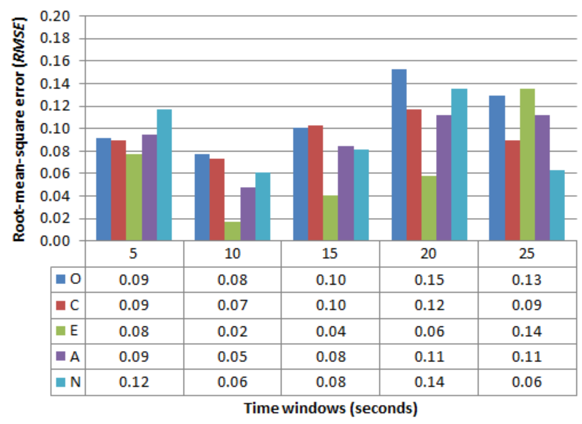
	5	10	15	20	25
🟦 O	0.90	0.96	0.91	0.88	0.90
🟥 C	0.90	0.93	0.90	0.89	0.91
🟩 E	0.93	0.98	0.97	0.80	0.77
🟪 A	0.94	0.99	0.97	0.96	0.97
🟦 N	0.98	0.99	0.95	0.94	0.96

Figure 6: Average correlation coefficient (*r*) for the Big Five factors (Openness (O), Conscientiousness (C), Extraversion (E), Agreeableness (A), and Neuroticism (N)) obtained for different time windows.



	5	10	15	20	25
🟦 O	0.81	0.93	0.89	0.95	0.81
🟥 C	0.91	0.93	0.83	0.80	0.78
🟩 E	0.88	0.96	0.88	0.81	0.98
🟪 A	0.89	0.99	0.97	0.96	0.97
🟦 N	0.79	0.98	0.96	0.90	0.87

Figure 7: Average coefficient of determination (R^2) for the Big Five factors (Openness (O), Conscientiousness (C), Extraversion (E), Agreeableness (A), and Neuroticism (N)) obtained for different time windows.



	5	10	15	20	25
🟦 O	0.09	0.08	0.10	0.15	0.13
🟥 C	0.09	0.07	0.10	0.12	0.09
🟩 E	0.08	0.02	0.04	0.06	0.14
🟪 A	0.09	0.05	0.08	0.11	0.11
🟦 N	0.12	0.06	0.08	0.14	0.06

Figure 8: Average root-mean-square error (*RMSE*) for the Big Five factors (Openness (O), Conscientiousness (C), Extraversion (E),

Agreeableness (A), and Neuroticism (N)) obtained for different time windows.

7 Concluding Remarks

In this paper we present a new method for interactive storytelling in games based on player behavior modeling, hierarchical task decomposition, and nondeterministic planning. The proposed method is capable of generating dynamic and nondeterministic quests that are directly or indirectly affected by in-game player behavior.

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 create their own narrative experiences.

As further research, we intend to conduct more tests in more complex game scenarios and genres. Also we plan to evaluate in depth a number of key aspects of the authoring process, especially authoring expressiveness, the complexity issues involved in the process, and the level of control the human author may have over the game's narrative. Furthermore, we consider that more extensive user studies are needed to evaluate our method from the player's perspective. Improvements to the behavioral model are also a paramount commitment in our research agenda.

References

[1] M. D. Back, S. C. Schmukle, B. Egloff. Predicting Actual Behavior From the Explicit and Implicit Self-Concept of Personality. *Journal of Personality and Social Psychology*, volume 97 (3), pages 533-548. American Psychological Association, 2009.

[2] S. C. Bakkes, P. H. Spronck, and G. van Lankveld. Player Behavioural Modelling for Video Games. *Entertainment Computing*, volume 3, pages 71-79. Elsevier, 2012.

[3] H. Barber, and D. Kudenko. A User Model for the Generation of Dilemma-Based Interactive Narratives. In *AIIDE 2007 Workshop on Optimizing Player Satisfaction*, pages 13-18. AAAI Press, 2007.

[4] R. Bartle. Hearts, clubs, diamonds, spades: Players who suit muds. *Journal of MUD Research*, volume 1 (1), 1996.

[5] C. M. Bateman, and R. Boon. 21st Century Game Design. Charles River Media Game Development, Cengage Learning, 2005.

[6] B. Bonet, and H. Geffner. Planning as Heuristic Search. *Artificial Intelligence*, volume 129 (1), pages 5-33. Elsevier, 2001.

[7] H. Borchani, G. Varando, C. Bielza, P. Larrañaga. A survey on multi-output regression. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, volume 5 (5), pages 216-233. John Wiley, 2015.

[8] P. T. Costa, and R. R. McCrae. Revised NEO Personality Inventory (NEO-PI-R) and NEO Five-Factor Inventory (NEO-FFI) professional manual. Psychological Assessment Resources, 1992.

[9] B. de Raad, and M. Perugini. Big Five assessment. Hogrefe & Huber, 2002.

[10] M. Ghallab, D. Nau, and P. Traverso. Automated Planning: Theory and Practice. Morgan Kaufmann Publishers, San Francisco, 2004.

[11] L. R. Goldberg. An alternative "description of personality": The Big-Five factor structure. *Journal of Personality and Social Psychology*, volume 59 (6), pages 1216-1229. APA Press, 1990.

[12] S. D. Gosling, P. J. Rentfrow, W. B. Swann. A very brief measure of the Big-Five personality domains. *Journal of Research in Personality*, volume 37 (6), pages 504-528. Elsevier, 2003.

[13] S. Haykin. Neural Networks and Learning Machines. Prentice Hall, 2008.

[14] U. Kuter, D. S. Nau, E. Reisner, and R.P. Goldman. Using classical planners to solve nondeterministic planning problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2008, pages 190-197. AAAI, 2008.

[15] R. Laws. Robin's Laws of Good Game Mastering. Steve Jackson, 2002.

[16] M. Lewis and K. Dill. Game AI Appreciation, Revisited. In S. Rabin, editor, Game IA Pro^2: Collected Wisdom of Game AI Professionals, pages 3-17. A K Peters/CRC Press, 2015.

[17] E. S. Lima, B. Feijó, and A. L. Furtado. Hierarchical Generation of Dynamic and Nondeterministic Quests in Games. In *Proceedings of the 11th International Conference on Advances in Computer Entertainment Technology*, Article N. 24. ACM, 2014.

[18] M. C. Machado, G. L. Pappa, and L. Chaimowicz. A Binary Classification Approach for Automatic Preference Modeling of Virtual Agents in Civilization IV. In *Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games*, pages 155-162, IEEE, 2012.

[19] T. Mahlman, A. Drachen, A. Canossa, J. Togelius, and G. N. Yannakakis. Predicting Player Behavior in Tomb Raider: Underworld. In *Proceedings of Computational Intelligence in Games*, pages 178-185. IEEE, 2010.

[20] B. K. Mishra. Psychology: The Study of Human Behaviour. PHI Learning, 2008.

[21] O. Missura and T. Gärtner. Player modeling for intelligent difficulty adjustment. In *Proceedings of the 12th International Conference on Discovery Science*, pages 197-211. Springer, 2009.

[22] I. B. Myers. The Myers-Briggs type indicator manual. The Educational Testing Service, Princeton, 1962.

[23] A. Nagle, P. Wolf, and R. Riener. Towards a system of customized video game mechanics based on player personality: Relating the Big Five personality traits with difficulty adaptation in a first-person shooter game. *Entertainment Computing*, volume 13, pages 10-24. Elsevier, 2016.

[24] S. V. Paunonen. Big Five Factors of Personality and Replicated Predictions of Behavior. *Journal of Personality and Social Psychology*, volume 84, pages 411-422. NCBI, 2003.

[25] A. Ramirez, and V. Bulitko. Automated planning and player modeling for interactive storytelling. *IEEE Transactions on Computational Intelligence and AI in Games*, volume 7 (4), pages 375-386. IEEE Press, 2015.

[26] M. Seif El-Nasr. Interaction, Narrative, and Drama Creating an Adaptive Interactive Narrative Using Performance Arts Theories. *Interaction Studies*, volume 8 (2). John Benjamins Publishing Company, 2007.

[27] M. Seif El-Nasr, A. Drachen, and A. Canossa, A. (eds.). Game Analytics: Maximizing the Value of Player Data. Springer-Verlag, London, 2013.

[28] M. Sharma, S. Ontañón, M. Mehta, and A. Ram. Drama Management and Player Modeling for Interactive Fiction Games. In *Computational Intelligence*, volume 26 (2), pages 183-211. Wiley Periodicals, 2010.

[29] P. H. M. Spronck, F. den Teuling. Player modeling in civilization IV. In *Proceedings of the Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 180-185. AAAI Press, 2010.

[30] F. Strack, and R. Deutsch. Reflective and impulsive determinants of social behavior. *Personality and Social Psychology Review*, volume 8, pages 220-247. Sage Press, 2004.

[31] D. Thue, V. Bulitko, M. Spetch, and E. Wasylishen. Interactive Storytelling: A Player Modelling Approach. In *Proceedings of the 3rd Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 43-48. AAAI Press, 2007.

[32] J. Tuunanen, and J. Hamari. Meta-Synthesis of Player Typologies. In *Proceedings of Nordic Digra 2012 Conference: Local and Global - Games in Culture and Society*, 2012.

[33] J. Valls-Vargas, S. Ontañón, and J. Zhu. Exploring Player Trace Segmentation for Dynamic Play Style Prediction. In *Proceedings of the Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 93-99. AAAI Press, 2015.

[34] G. Van Lankveld, P.H.M. Spronck, H.J. Van den Herik, A. Arntz. Games as personality profiling tools. In *Proceedings of the 2011 IEEE Conference on Computational Intelligence in Games*, pages 197-202. IEEE Press, 2011.

[35] B. Weber and M. Mateas. A Data Mining Approach to Strategy Prediction. In *2009 IEEE Symposium on Computational Intelligence in Games* (Milano, Italy), pages 140-147. IEEE, 2009.

[36] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. André. Player Modeling. In *Artificial and Computational Intelligence in Games*, pages 45-55. Dagstuhl Publishing, Saarbrücken Wadern:, 2013.