ADAPTIVE STORYTELLING BASED ON MODEL-CHECKING APPROACHES

Nicolas Rempulski and Armelle Prigent and Pascal Estrallier and Vincent Courboulay and Matthieu Perreira
Computer Science Laboratory (L3I)
University of La Rochelle
E-mail: nicolas.rempulski@univ-lr.fr

KEYWORDS
Interactive storytelling, video game, game design, adaptive method, model-checking, formal model.

ABSTRACT
Interactive scenarios are often represented as branching-like systems. This representation complexity is increased as options left to players grow. Each fork in the story force designers to define all possible choices and their following situations.

We propose a formal model of scenario to ease designers task while granting the same freedom to players. In this model each entity is modeled as an automaton. These automata are then combined to build the scenario as all entity's actions sequences. The right sequence is then unfolded on line, while the player experience the story. The structure of the narration is guaranteed by properties verification on the model. These properties are set by designers to provide an interesting narrative frame to players.

INTRODUCTION

All media are not able to propose interactive storytelling. Indeed altering the story, or building it while relating, imply to modify the media itself. Video games as a numerical medium easily permit such modifications. Scriptwriters can thus create stories allowing players a great immersion by granting them a role in the story unfolding. Classical storytelling is based on linear narrative. The addressee undergoes the scenario and is not consulted to progress in the story. For instance books, films, and some video games use linear narratives.

However linear narration is more secure for designers. Indeed they can guarantee stories' structure and quality since they know the situations, and their sequence, that the player will face. Furthermore authors can use classical narrative structures to produce emotions known to the player. Joseph Campbell in “hero with thousands faces” proposes a typical sequence of actions that is common to many stories. This work has been used by Christopher Vogler to analyse Disney animation films.

Interactive Storytelling

The interactive storytelling approach is a more complicated way. Jenkins has said that in the video game framework, “the game designer is a narrative architect” (Jenkins 2003). Here, the designer proposes a frame for the story to unfold. He has to consider that the player will be the narrator as he/she chooses actions to be carried out. This means that the final addressee of the narrative, here the player, will realize the final narrative act. Whatever the proposed storytelling kind, the player will make some choices that could attempt the logic of this narrative.

A possibility to resolve this problem is to constrain the player choices at some points of the narrative. As the designer wants to lead him/her in a more specific way in the story, the interactivity can be restricted to required player's actions. The major problem is that the player can become frustrated when he/she is not be able to make real choices in the story.

Thus, in this restrictive narrative case, the player choices are limited:
• He/She is not allowed to make decisions (even if he/she believes the opposite). The player will become frustrated when he/she becomes aware of it;
• Game's endings are planned by the designer;
• The player tries to transgress in order to explore other possibilities. This search is often ended with a failure.

On the other hand, if the player is completely free in his/her actions, he/she can corrupt narrative logic. During the game, the player has the following choices:
  • Do nothing;
  • Make bad choices (transgression, error);
  • Test impossible actions (digression);
  • Anticipate actions (disordering narrative);
  • Make good choices (the designer's ones) if he/she wants to finish the game.

The designer has to find the balance between the freedom left to players and their constraints in order to guarantee the unfolded story's quality. Classical interactive storytelling approaches are based on branching scenarios. Thus, increasing the player's freedom consists of increasing the possible paths set at each decision point. For instance, in the game Façade, the whole storytelling is adapted to the player choices (Mateas 2002). The player is invited to dinner with a couple. The player has to interact with two characters who just celebrated their tenth wedding anniversary. Depending on the player's choices, this couple can breakup at the end, be more in love than ever and have a variety of other endings between these extremes. In this approach, thousands of pre-written scenario segments have been produced. This is a branching scenario based on a thousand branches which is an expensive amount of work to build and verify.

In branching-like scenario, finding the balance between freedom and control is tortuous work for the designer. Indeed, the designer has to take into account all possible choices of the player and define their implications. In our approach, this balance is calibrated “on line”, i.e. while the story is built. The designer can propose interactivity and freedom to the player while being sure that defined structure and the meaning of the story are preserved.

Story structure

To control the story's structure we have chosen to guarantee its dramatic quality. Aristotle wrote the dramatic arc which is a generic analysis of tragedies' structure. He describes the dramatic arc as lifting the audience to an emotional climax then resolving with catharsis. This dramatic rule is the scenario's structural rule established by the designer. No matter what happens while the story unfolds, this rule respects the dramatic structure called dramatic intensity. We use formal methods and model verifications, called model-checking during the game execution to ensure the dramatic intensity.

Formal Method

Formal methods are useful to analyse the game's dramatic intensity quality during its execution. Our approach consists in representing the story by a formal model based on automata network. This model, which is an abstraction of the story and its state, is analysed on-line in order to guarantee that the dramatic intensity evolution is strong. The analysis method called model-checking is classically used for checking critical systems. The model-checking method consists in verifying model quality with respect to a set of properties. The global system states space which represents all possible paths along the scenario, is generated and each state is verified to ensure whether or not they respect the properties. Using these verifications, an expert system dynamically modifies the scenario using properties based on dramatic intensity. The chosen formal model is a timed automata network and the verification is realized using the model checker Uppaal (Larsen and Al. 1997).

Related works

The “Centre for Virtual Environment” of the University of Salford works on scenario analysis in pedagogical tools and games framework. This work is focused on the unfolded story analysis. For instance, in the educational tools framework, it is important “to force” the user to follow a specific scenario (to make sure that he/she has met all important information or to carry out evaluations). However, if the user has a small amount of freedom on his/her tool exploration, the immersion will be weak and its attraction for the tool will decrease (especially in the games framework). Their problem thus consists in being able to define a scenario which can be controlled while offering a great freedom to users. Thus they use an
“emergent narrative” concept. This concept is based on an improvisation model rather than on a scenario-based model.

The Liquid Narrative group proposes an architecture based on a system expert connected to an unreal engine to manage narrative (Young and Al. 2003). ID-tension is another project which relates to the dramatic intensity control of a story and creates the concept of agent narrator. The purpose is to build a generator of interoperating stories (Szilas 1999). Its interactive drama model is composed of the following elements:

- The world in the story represents the current world's state;
- Narrative logic includes the story's evolution rules meaning how to connect actions;
- Narrator decides actions that will be proposed to the user;
- The user model describes the user's characteristics and allows the narrator to determine which actions to carry out;
- Theatre allows interaction with user (HMI).

The Cedric team (Vega and Natkin 2003) proposes an approach for game analysis. Their work concerns video game analysis and specification by proposing a method which guarantees that actions carried out by the player remain coherent throughout the story. With this intention, they use Petri's nets subnet construction techniques. The method guarantees that if each subnet is coherent, the unit built by respecting a set of rules remains coherent.

In (Champagnat and Al. 2005), the scenario is modeled by linear logic and an adaptive architecture to compute some verification of specific properties in order to drive the storytelling.

Outlines

Our approach has been tested on an interactive game called L3ILife developed to test interactive systems. Our current prototype works with the presented scenario model and model-checking method to dynamically modify the game. Finally, we are exploring techniques to analyse player behaviour and adapt the scenario dynamically.

MODEL CHECKING FOR INTERACTIVITY

This section introduces our method for controlling dramatic intensity during the game session by using dynamic verifications. We have to write the scenario in a specific way in order to be able to perform dynamic verifications of the dramatic intensity's quality. The scenario writing with this method consists of three steps:

- First, each game's entity is modeled as an automaton describing its possible states and actions. These actions represent scenario's evolutions that this entity can trigger.
- Each entities' action is weighed by a dramatic intensity modification. From the player's point of view, if it's a good action (leading to a favourable end) dramatic intensity decreases otherwise it increases.
- Finally, desired dramatic intensity evolution during execution has to be defined. This is called the dramatic intensity curve.

Our interactive system is in charge of adjusting a game's events in respect to the dramatic intensity curve during a game session. To do this, our expert system will choose, over an already defined controllable actions set, the right action that will satisfy the expected dramatic intensity evolution. This method can appear as non-deterministic. Indeed, a scenario writer will possibly want “to force” players to meet some specific events in the story. However, it is possible to specify behaviours composed of set atomic actions which players will meet in the story.

We will first present our interactive game prototype.

Prototype of Interactive Game

To validate our method consistency, we have developed a prototype called L3ILife. Built with Unreal Engine it models realistically L3I laboratory's buildings. The player embodies a student who has to find the CD containing his/her work, which is spawned randomly on the map. Once he/she has explored offices and class rooms and found his/her CD, he/she has to print it on one of the three existing printers. Once his/her documents are printed he has to give them to his/her teacher. The player has a
caffeine gauge that decreases each second. He/She can pick up coffee to increase his/her caffeine level. If the gauge reaches 0 the game is over.

Two other students (bots) attempt to place hindrances on the player. Claire has the same goal as the player. She will try to give her own work to the teacher and prevent the player from succeeding. For example, by destroying his/her documents or CD. Paul is trying to prevent the player from giving his/her work to the teacher. His goal is to catch him/her and if successful, the game is over.

The first iteration of this game offered one interface for the player, the first person view, and one for the “game master”. This one can control game state and launch game event. For instance he/she can order Claire to destroy the player's CD or place some printers off line.

In this iteration, and with the interactive storytelling automation, game commands have been transferred to an expert system which we will describe later.

In the following subsection, we will present the scenario model which we use to define how to unfold interactive story.

**Controlled Scenario Model**

As our approach is based on formal method, here we define the formal elements used to model a scenario.

---

**Fig 1: Screenshot of L3ILife prototype**

---

**Quantified scenario Model**

**Definition 1 (Constraints on variables):**

An atomic constraint compares a single variable value with a constant. A constraint is a conjunction of atomic constraints.

The set $\phi(V)$ is the constraint set on variables and is defined in the following way:

$$\phi := x \leq c \mid c \leq x \mid x < c \mid c < x \mid \phi_1 \land \phi_2$$

where “$x$” is an integer variable and “$c$” is a constant.

We introduce the concepts of “controllable” and “uncontrollable” actions from the point of view of the expert system.

**Definition 2 (Controllable actions):**

The set of actions in the game is divided into two sets. The controllable ones are actions that are decided and executed by the expert system such as bots orders and resources spawns. The uncontrollable actions are the ones performed by players.

Controllable actions are triggered to make the story evolve. When a game's quality is no longer respected, the expert system will be able to correct the dramatic intensity with these actions.

A narration's quality is guaranteed by respecting properties on dramatic intensity. We need to increase classical automata with this particular variable, named quantifier. Each automaton represents an entity's state. Evolution from one state to another is directed by a transition.

**Definition 3 (Quantified automata):**

The quantified automaton is an extension of classical finite state automaton. It is defined by a $n$-tuple $(L, l_0, \Sigma, V, Q, \rightarrow)$ where:

- $L$ is the set of automaton's states
- $l_0$ is the initial state;
- $\Sigma$ is the set of synchronisation labels. This set is formed by controllable and uncontrollable synchronisations; $\Sigma = \Sigma_c \times \Sigma_u$. The synchronous communication between processes is done by “GO”. The alphabet $\Sigma$ is made of a labelled set of reception synchronisation noted as “?”, and emission synchronisation noted as “!”. Transitions without synchronisation are labelled “$\tau$”;
- $V$ is the set of integer and boolean variables of the model;
• Q is the quantifiers set of the model related to the properties expression;
• → is the set of transitions as:
  \[ \rightarrow = \mathcal{L} \times \Sigma \times \mathcal{B} (Q \times V) \times 2^{Q \times V} \times \mathcal{L}. \]

Each transition is tagged with a synchronisation, a constraint on quantifiers or model variables and one or more assignments on quantifier or variables.

\( \mathcal{B} (Q \times V) \) represents the constraints on transitions that made it executable (see definition 1). \( 2^{Q \times V} \) is the set of assignments of variables during the transition execution.

The automaton on figure 2 is a quantified automaton representing Paul's behaviour in the game. Each action carried out by this Non Player Character (NPC) can increase or decrease the dramatic intensity quantifier, named drama in figure 2.

**Definition 4 (Automata network):**

Automata network is defined as a parallel composition of processes \( S_1, \ldots, S_n \), combined in only one system by the CCS parallel composition operator. This system must be closed, i.e. it has to prohibit external events occurrence. A network state is a couple \( (L, U) \) where \( L \) is a network’s subsystem state vector and \( U \) a valuations vector of the set of variables \( V \times Q \).

The quantified scenario model is a network of quantified automata of each entity in the game. The player behaviour is modeled by a specific quantified automaton representing all possible actions of the player.

**Definition 5 (Controlled scenario model):**

The executed model is a quantified and controlled automata network. It is defined by a n-tuple \( (A_E, C_A, A_J, S_J) \) with:

- \( A_E \) is the set of a game's entities automata;
- \( C_A \) is the set of entities' decision automata;
- \( A_J \) and \( S_J \) respectively represent the player's automaton and the associated decision automaton.

We thus built various automata in a wide finished states representation:

- **Quantified scenario model:** Two automata for NPC's actions (Paul and Claire), and an automaton for player’s actions (see figure 3 left part);
- **Decision model:** An automaton standing for a player’s decisions and an automaton modeling an expert system's decisions (see figure 3 right part).

The CSM (Controlled Scenario Model) is an automata network whose execution is similar to automata in finished states systems. The system is known as closed since the synchronisations set and received from transitions is associated with emissions in another automata of the model. The total system execution will thus behave according to the traditional definition of the decisions that this entity can take, marked as available transitions. An additional quantified automaton represents a player’s actions, this automata is also linked with a supervisor, making it possible to simulate the player's decision making process.

From now on, the scenario dynamic adaptation is based on a model representing all possible actions in the game in any order. This model is dynamically analysed to determine the action which the NPC must carry out.

This controlled scenario model consists in combining two different models:

- A scenario model that represents the set of game's entities and their possible evolutions
- A decision model that represents the decisions of these entities, handled by the interactivity engine, as well as their availability. These decisions can be taken by the player himself/herself or by the expert system for NPC.

**Fig 2. Paul's quantified automaton**

**Decision Model**

In order to be controlled, the scenario model is enhanced by additional automata carrying out the supervision.

Each automaton, linked to a game entity, is associated with a local supervisor. These represent
synchronized automata network. A state of the automata resulting from the composition corresponds to a vector containing the current state of each subsystem.

![Diagram of automata network](image_url)

**Figure 3: Interaction between automata in the model**

Figure 3 shows synchronisations between A_i's automaton representing player's actions in the scenario, and S_j's automaton, used to simulate decisions taken by the player. In the top part of the figure 3, we can see all the scenario model automata for game entities (A_E) and their decision model associated (C_A).

**INTERACTIVE MODEL AND VERIFICATION**

From the controlled scenario model definition, we extend the model used to dynamically analyse the scenario quality.

**Definition 5 (Model of adaptive game)**:
The adaptive game model is a pair (Sc, Fe) where Sc is the CSM, such as we defined it previously, and Fe is a function of expected evolution of quantifier in the CSM.

**Dramatic intensity evolution**

The driving scenario is carried out by a dynamic analysis of the adaptive game model defined previously. This analysis consists of a forward reachability analysis using the model-checking tool, Uppaal.

Our main objective, in this adaptive system implementation, is to guarantee the scenario quality during a game session. This quality can only be evaluated on discrete elements, since most of a player’s experiment parameters are subjective to an adaptability engine. Thus the stress which the player feels, while experiencing the game’s scenario, is a good starting point. If the player is immediately subjected to a very stressful environment, it will be difficult to increase this stress. That can lead to an early defeat of the player, who does not have time to get his/her bearings, or worse, make him/her have a lack of interest for the story.

Many parameters related to the quality of this scenario can be analysed during the game. For instance, it is possible to quantify players' skills to evaluate the scenario's difficulty to produce. When designing this prototype, we focused on one factor: stress peaks and the difficulty that they cause. Among the elements of evaluation which we previously quoted, we thus chose to use the dramatic intensity to monitor both a player's actions and a game's state. Dramatic intensity corresponds to the stress level to which the player is subjected. Each negative event, in which it makes it more difficult for the player progress, increases the dramatic intensity. Conversely, any event advancing the story towards an outcome in favour of the player, causes the dramatic intensity to drop.

**Model checking properties**

The approach consists in evaluating a simple property on the CMS. Indeed, as we want the dramatic intensity to reach a specific value at some points of a game's execution, we use the reachability property analysis. We will first introduce some concepts about the temporal logic used by the model-checker to perform analysis.

**Temporal logic properties**

Uppaal makes it possible to verify properties, expressed in temporal logic CTL, by methods of model-checking.

**Definition 6 (CTL property)**:
CTL properties are made according to the following grammar:

\[ \varphi ::= A \square \beta | E \diamond \beta \]

where

\[ \beta ::= a | \beta_1 \land \beta_2 | \neg \beta. \]

“a” is an atomic formula which can be either clock constraint or state component.

Reachability properties

Reachability properties consist in expressing a formula of state that can be checked out during the system execution. Another wording is to ask the question: “From a given system’s state, is there a path to another system’s state where this property is satisfied?” Reachability properties are written as:

- \( E \diamond s \) to find a path to reach the state \( s \);
- \( E \diamond v=0 \) to find a path to reach a state where the variable \( v \) is equal to 0.

We use the second one to express the property that allows us to reach a state where the quantifier is at a given value.

The counter-example trace is produced by Uppaal’s tool of analysis, called verityta, which carries out an analysis of reachability on our controlled scenario model expressed in the form of automata networks.

Counter-example trace

However, the model-checker generates a counter-example trace only if the property is not verified. We then have to provide the negation of our reachability property to obtain a trace for this property.

Safety properties are a negation of reachability properties. Thus the verified property becomes: “From a given state, there is no path to another system state where this property is satisfied” written as \( A[\text{v!}=0] \). We use the model-checker to ensure that there is no path to a state where the quantifier is at the wanted value.

The anticipated answer is that this property is not verified. Indeed we have built a model and drama intensity curve as to always find a path. Then it provides us a counter-example trace to show us why this property is not satisfied. This path is a succession of events produced to reach the wanted drama intensity value.

Each time the system detects that the drama intensity is at a wrong value, according to the drama intensity curve, it launches a verification to find the path which unfolds to reach the next drama intensity step.

ADAPTIVE ARCHITECTURE

In this section, we present the architecture implemented for the dynamic scenario control and the dramatic intensity level management. It has been tested in our prototype of the 3D adventure game L3ILife.

Architecture

Our architecture is based on a multi-agent system. It is organized around three principal agents (see figure 5):

- The observer agent is connected with the system to observe, it transmits games’ events to the scenario writer agent.
- The scriptwriter agent analyses events and models and sends relevant new actions to the agent director.
- The director agent executes new actions in the game system.

This architecture manages two models. First, the dramatic intensity curve contains the state of dramatic intensity with respect of time written by the designer. The second one is the Uppaal model that represents all possible actions in the game (CMS).

All analysis complexity is thus carried out by the scriptwriter agent. This one, thanks to the events provided by the observer agent and a model of
possible scenarios, will choose the scenario that makes it possible to reach the desired dramatic intensity. It will then transmit events to the director agent.

The script writer agent is not able to decide uncontrollable actions, it must choose from the set of controllable events. Thanks to the event based communication, the set of events that can be handled by the observer agent can be easily extended. It can for example receive information concerning the player behavior. Such an extension is described in the following section.

DETECTING PLAYER BEHAVIOR

In previous work (Perreira and Al 2007) we have described a new kind of gameplay based on gaze tracking. One of our objectives was to detect the player’s behavior in order to adapt dynamically the scenario using our adaptive architecture. Technical details about the gaze tracking system can be found in (Perreira and Al 2008).

During the first experiments, we only took into account a few explicit players’ behavior: Firstly, we focused on the interaction with the non-player character named Claire. She tries to steal the work of the player. The player can interact with the girl by doing a wink to the camera when he/she is in front of the girl. Then, Claire will give her work back to the player if she has stolen it to him/her.

Secondly, we allow the player to protect himself/herself against the evil student by looking down (see figure 6). Indeed, if Paul arrives near the player, this one will put his head down and then the evil student will go on without stealing his/her work.

We have observed that this new kind of interaction improves the players’ immersion in the virtual world of the game. Moreover, it also increases its interest for the game because the gameplay is
richer and the game more fun to play. These observations will certainly be confirmed once our framework will include the observation of implicit behavior as we will have more real-time feedback on the user's gaming experience.

We are currently working on the integration of more complex kinds of behavior into our interactive game framework. Gaze tracking could be used to observe the level of attention of the player. For example, if the player stops watching at the screen, a particular game action can be launched to refocus his/her attention. In L3iLife for example, in this kind of situation, the adaptive architecture can modify the unfolding of events, and make the evil student run after the player to steal his/her work. It is a stressing action that can bring some interest back to the player.

This implicit behavior system is not yet implemented in L3iLife, but we intend to. One of our future goals is to show how adaptive storytelling is able to use the player’s implicit behavior to increase game immersion.

CONCLUSION

In this paper we have presented a method that uses on the fly model-checking approaches for driving storytelling throughout a video game. This method allows the management of a dramatic intensity curve during unfolding of a game.

We have implemented the architecture in a multi-agent system and connected it with Unreal Engine. The dramatic intensity curve is correctly managed during the game and the system is able to modify the scenario during the game.

From this approach, we intend to extend this method by driving other parameters during the game such as difficulty, challenges and local and global time execution and use gaze tracking to adapt the scenario to the implicit behavior of the player.

REFERENCES


BIOGRAPHY

Nicolas Rempulski is a Phd student at the L3I computer science laboratory. After a master thesis in 2008 on dynamic control of narrative, he is beginning his Phd on the same research area.

He has participated in the development of an ubiquitous game demonstration project at the eArts festival 2008 in Shanghai.
Armelle Prigent is an associate professor at L3i laboratory. She received her PhD in computer science in 2003. During her doctoral work, she proposed a method for the parameterized test of real time systems. Her research interests are now on formal approaches for the driving of interactive application. She is co-head of a research team on interactivity in L3i laboratory and teacher at video game school (ENJMIN) in Angoulême.

Pascal Estrallier is a full professor in the Computer Sciences department at the University of La Rochelle. His research concerns architecture of interactive software, game design and the new gameplay methods. He has headed the L3i Research Laboratory Informatics, Image, Interactions to which 67 researchers are affiliated. Moreover, he is, in shared time, project manager in the department of Information Technologies of the French Ministry of research, in charge of Computer Science area. He is co-founder of the first French Educational program on Games at Master Level (ENJMIN)