In 1st Joint International Conference on Interactive Digital Storytelling, Erfurt, Germany

Developing a Drama Management Architecture for Interactive Fiction Games

Santiago Ontañón, Abhishek Jain, Manish Mehta, and Ashwin Ram

Cognitive Computing Lab (CCL)
College of Computing, Georgia Institute of Technology
Atlanta, GA 30332/0280
{santi ,abhishek, mehtama1, ashwin}@cc.gatech.edu

Abstract. A growing research community is working towards employing drama management components in interactive story-based games. These components gently guide the story towards a narrative arc that improves the player's experience. In this paper we present our Drama Management architecture for real-time interactive story games that has been connected to a real graphical interactive story based on the Anchorhead game. We also report on the natural language understanding system that has been incorporated in the system and report on a user study with an implementation of our DM architecture.

1 Introduction

There has been a growing interest in creating story-based interactive fiction games where the player is considered an active participant in the ongoing narratives. The component in charge of guiding the complete dramatic experience is called Drama Manager (DM) [2] or Director [4]. The DM employs a set of actions provided at appropriate points in the ongoing game whereby the player is guided towards certain aspects of the story. Previous approaches to drama management have either not been connected to a concrete world [11] or have been evaluated without using real human players interacting with a real game [7]. In our previous work [9] we evaluated search based DM techniques in a simple implementation of the game Anchorhead [3]. Anchorhead, created by Michael S. Gentry, is a game with a complicated story and several subplots, making it amenable for drama management studies.

Previously, we have shown that player modeling is a key component for the success of drama management based approaches [8] and DM should take into account player's previous gaming experience and player experience with the current game in its use of strategies. We have also reported [9] that the player model should not only be built using the feedback on interestingness of intermediate game events but also on the player feedback on various strategies used by DM that were visible to the player during the interaction. In this paper we will focus on three main goals. First, we tackle the problem of using search-based DM techniques in real-time games. The vast majority of commercial interactive stories

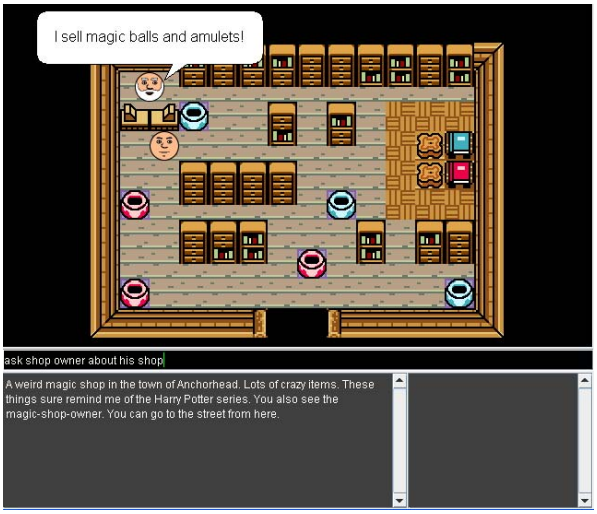

Fig. 1. A screenshot of our graphical implementation of Anchorhead.

are real-time, so it is very important to take this issue into account. Second, most search-based DM techniques assume a limited set of actions the player can execute (in order to cut down the size of the search space that the DM has to explore), in this paper we will explore how to apply DM techniques to a game where the number of actions the player can execute is increased as a natural language interface (in our game) creates a more open ended input action space. Finally, we report a user study with an implementation of our DM architecture and an initial analysis of the study. This report is useful in two ways: a) it supports that our DM architecture is effective in real-time games, with open ended user actions, and b) provides guide for future development of our research.

The rest of the paper we will first present a brief introduction to the game we have used as our testbed, Anchorhead and present our interactive stories architecture. After that, we introduce our technical approaches to NLU, player modeling and drama management, followed by the results from the player evaluations. Finally, we conclude the paper with some final thoughts and future directions.

2 Interactive Stories Architecture

Anchorhead is an interactive story game created by Michael S. Gentry [3]. We have developed a subset of the complete game and it includes a graphical interface for interaction with the player. Graphical as well as text descriptions of the current scenario are presented to the player, who then enters commands in textual format, e.g. “enter the mansion” or “take the key”. For this paper we have focused on a subpart of the story, identified by [7] as interesting for evaluating drama management approaches. Figure 1 shows a screenshot of our Anchorhead

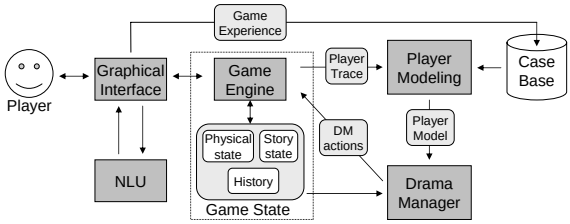

Fig. 2. Overview of the main modules and their relations in our interactive story architecture.

implementation. To evaluate our approach we developed a generic interactive stories architecture and implemented a graphical version of Anchorhead with it. Our architecture consists of five modules (shown in Figure 2), namely:

- Graphical Interface (GUI): through which the user interacts with the system, shows a graphical representation of the game (see Figure 1), and allows the player to enter commands in English.
- Natural Language Understanding (NLU): parses the English text and generates a representation that can be understood by the game engine.
- Game Engine (GE): responsible for running the game, maintaining the physical state, story state and a history of what happened during the game.
- Player Modeling (PMM): develops a player model using case-based reasoning techniques from the player actions.
- Drama Manager (DMM): takes the player model and the current game state and generates drama manager actions (DM actions) in order to influence the course of the game towards more interesting plots for the current player.

All the modules in our architecture are developed independently and are domain independent. In order to define a game, the author has to specify (using XML files) the initial state, graphics, the set of plot points and the set of DM actions available.

A game executes in the following way. When the system is launched, the player specifies his previous playing experience (in a 5 point scale) and the number of previous times he has played the game (this is used as part of the player model). Then the player starts playing the game. Each time the player enters an action, the current player trace is updated and the player modeling module updates the player model. The drama manager runs in parallel with the game and constantly is performing time-bounded searches for appropriate DM actions to execute. Each time the DM finishes a search, it executes the resulting DM action, and each time the player performs an action, the DM checks whether the action invalidates its search process or not, and restarts the process appropriately. When the game is over, the player is presented with a feedback form, where he can specify which parts of the game he enjoyed and which parts he didn't. The form allows the player to provide feedback for plotpoints as well as for observable drama manager actions (like hints). The result of the form is stored in the case base of the player modeling module for future games.

NLU Submodule	Submodule Output
User Input	Could you take me to the living room
Semantic Analyzer	<action_name=go_to> <attribute name=location value=livingroom> <attribute name=dialogact value=question>
User Input	give me the flask please i need it
Semantic Analyzer	<action_name=give> <attribute name=topic value=player> <attribute name=object value=flask>

Table 1. *Two Examples of processing inside the NLU.*

2.1 Natural Language Understanding

The text typed by the user is sent to the NLU module where it creates a semantic representation for the input that can be further used by the game engine. The NLU approach is based on techniques used previously in [5, 6]. The NLU consists of two main parts: a key phrase spotter and a semantic analyzer. The first stage of processing inside the NLU involves detecting multi-word expressions from a stored set of words labeled with semantic and syntactic tags. After detecting the keyphrases from the user utterance, the processed utterance is sent to the semantic analyzer for further processing. Here, dates, age, and numerals in the user utterance are detected while both the syntactic and semantic categories for single words are retrieved from a lexicon. Relying upon these semantic and syntactic categories, grammar rules are then applied to the utterance to help in performing word sense disambiguation and to create a sequence of semantic and syntactic categories. At the same time, the NLU calculates a representation of the user utterance in terms of dialog acts. Generic rules are defined inside the semantic analyzer for detecting dialog acts. These dialog acts provide a representation of user intent like types of question asked (e.g., asking about a particular place or a particular reason), opinion statements (like positive, negative or generic comments), greetings (opening, closing). The final output is a semantic representation consists of the action that the player wants to conduct in the game and various properties of the actions encoded as attribute-value pairs. Attributes of the actions are things like the location at which the action needs to be conducted, object to which the action should be applied and so on. Table 1 provides two examples of processing inside the NLU.

2.2 Player Modeling Module

The player modeling module (PMM) maintains a player model for the current player based on the feedback provided by players at the end of each game. This feedback contains player opinions on the game, including the parts they enjoyed and those that were not interesting from their perspective. At the end of each interaction, the PMM stores this player feedback along with the corresponding trace of player actions during the game. When a new player interacts with the

game, the stored feedback of previous players is used to predict which storylines and game elements would be enjoyable for the new player.

Specifically, before starting the game, the player enters his overall gaming experience and the number of times he has played the Anchorhead game, and at the end of each game, the player is presented with a sequence of the plot points that he visited over the course of the entire game episode. From the list, the player is asked to select his preference of the plot points based on a 5 point Likert scale: *strongly like*, *like*, *indifferent*, *dislike* and *strongly dislike*. After that, the player is presented with the list of observable DM interventions (mostly hints) that occurred during the game. The player rates them on a 5 points scale as well. Finally, the player rates the game as a whole.

We use a case-based reasoning (CBR) approach [1] for the PMM module. Based on the feedback provided by the player, the systems builds a *player preference model* that models the stories and DM actions that the player is likely to enjoy. CBR works by reusing previous experiences (called cases) to solve new problems. In our CBR approach for player modeling each time a player plays a game and provides feedback, the trace of the game plus the feedback is stored in the form of a *case*. As a particular player is playing the game, his current trace is compared to the traces in the different cases stored in the case base. The PMM retrieves the most similar traces and creates a player model by aggregating the player feedback contained in them. The assumption made here is that the trace of a player captures its playing patterns, and that players with similar playing patterns will have similar preferences. Specifically, cases in the PMM consist of the following elements: a) Player gaming experience, on a 5 point scale; b) number of times the player had played Anchorhead before; c) the game trace, consisting of a sequence with single action that the player executed, and each DM action that the DM executed in the game; and d) all the information provided by the player in the feedback form: overall experience rating, confidence, plot-point preference and DM action preference.

The two key processes in the PMM are: a) how to compare the trace of the current player with the traces stored in the cases in the case base, and b) once a subset of cases have been identified as relevant, how are they used to generate a player model. Let us explain those two processes in detail.

Case Retrieval Case retrieval works by computing a distance metric among the current player and previous players (stored as cases) and selecting a subset of k cases with the minimum distance. In order to compute such distance metric, the PMM uses 4 different distance metrics, that are averaged in order to compute the final distance: a) difference in player gaming experience, b) difference in the number of times the player has played the game before, c) difference of the player action traces, and d) difference of the DM action traces.

Each of the distance metrics generates a number between 0 and 1. The metrics for gaming experience and number of times the player has played the game are simple, so let us explain how do we compute distances among traces.

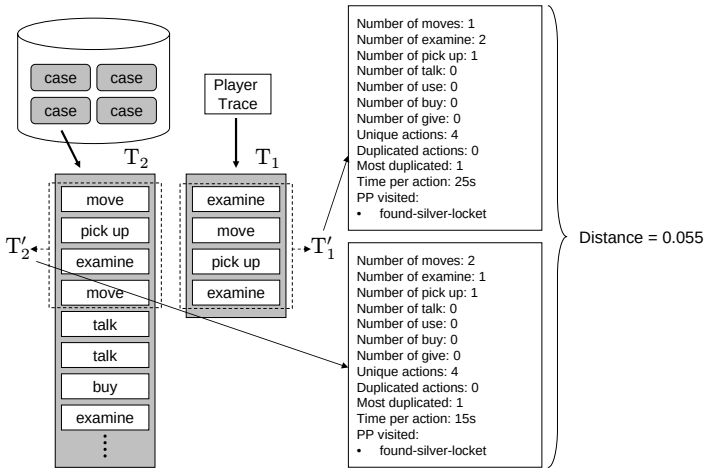

Fig. 3. Computation of player action trace distance.

To compute how different a player action trace is from another we perform some preprocessing (as shown in Figure 3). Specifically, if we are comparing a trace T_1 with a trace T_2, having n_1 and n_2 actions respectively, we only consider the first $min(n_1, n_2)$ actions of each trace. Let's call these reduced traces T'_1 and T'_2. Now, T'_1 and T'_2 have the same number of actions and can be properly compared. The rationale behind this process is that when a player has just started to play the game and has performed only a few actions, it does not make sense to compare its trace against the full trace of another player.

From T'_1 and T'_2, we compute a set of features consisting on: the number of different kinds of actions (movement, talking, using, etc.), number of unique actions, number of duplicated actions, the highest number of times a player has repeated an action, the average time a player takes to decide an action, and the difference among the set of plot points explored in T'_1 and T'_2. Each of those features define a number, and to compare two traces we simply compute a normalized Euclidean distance, which generates a number between 0 and 1.

In order to compare two DM action traces, we perform the same preprocessing as for player action traces and obtain the reduced DM traces, and define an analogous set of features.

Player Model Generation The output of the retrieval process is a subset of k cases ($k = 3$ in our experiments) relevant to the current player: $c_1, ..., c_k$, and their distances to the current player $d_1, ..., d_k$. A player model in our system consists only of three elements: a) a list of predicted plot point interestingness, b) a list of predicted DM action interestingness, and c) a confidence value in the interval $[0, 1]$, corresponding to how sure the PMM is that the current module is correct. To build such model, the interestingness value for each plot point, and DM action is computed by aggregating the interestingness values from cases by

using the similarity metrics to weight the individual interestingness values, i.e. the interestingness values in case c_i have d_i as their weight. The confidence is computed in the same way by averaging the confidences in the cases weighting them by their distances. The output of the PMM is a player model that consists of the predicted interestingness of each plot point for the current player and also a confidence on this player model.

2.3 Drama Management Module

Given the player preference model, the current game state, and optionally some author specified story guidelines, the Drama Management Module (DMM) plans story arcs that maximize both the player model and the author specified story guidelines. The DMM module constantly starts searches in parallel with the game using this information to select, if necessary, a particular action to influence the story towards the story arc identified by the DMM as the most interesting. The game author specifies a set of *drama manager actions* (DM actions). These actions represent the things that the drama manager can carry out to influence the game, e.g. "prevent the player from entering the library by locking the door" or "make the bar keeper start a conversation with the player about the suspicious person". The DM actions can be classified in several groups:

- *hints*: actions that has no direct effect on the game, but hint the player towards a particular direction.
- *causers*: forces something to happen in the game instead of waiting for the player to do it.
- *deniers*: prevents the player from doing something.
- *temporary deniers*: only prevent the player from doing something temporally.
- *reenablers*: reenables a previously denied line of action.

Notice that some of the actions (such as deniers or causers) have to be subtle so that the player does not feel manipulated.

Each DM action consists of several elements: a name, the type, a set of preconditions, and a set of effects. The preconditions of a DM action are boolean expressions that can test whether some plotpoints have already been visited, if the player is in any particular location, or has a particular object, etc. The effects that a DM action can have are: causing characters in the game to execute actions, firing plotpoints, directly modifying the game state, or hinting actions.

An example of DM action in our implementation is "bum_hints_crypt_key_is_in_basement", that has as preconditions that the player is in the park, that the player has bribed the bum and that the player does not have the crypt key. The effects of the actions is causing a particular character in the game, the bum, to tell the player that there is a key hidden in the basement. Specifically, this action is a *hint*. As the game is going on, the DM will choose to execute this action if it realizes that by providing the key to the player, it will cause the player to reach enjoyable plot points.

The algorithm used the the DMM is a variation of the algorithm explained in [9], based in an *expectimax* method, but adapted to real-time games in which the player can type actions through natural language.

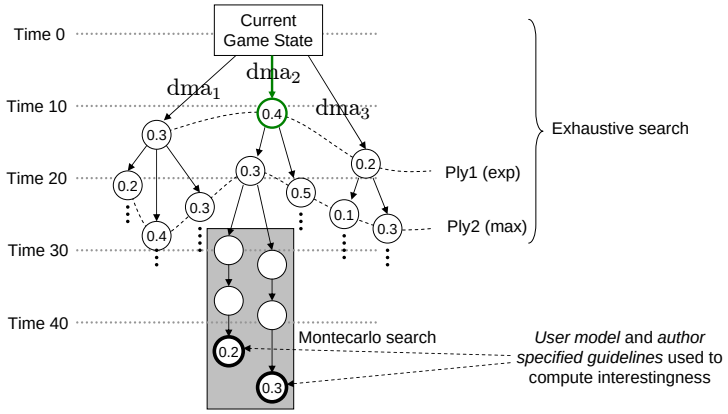

Fig. 4. Visualization of the search process performed by the DMM, where the DMM had to decide among three DM actions.

Searching For an Interesting Story Arc In order to decide which DM actions to execute, the DMM performs a search in the space of possible stories. Such search is represented as a search tree with alternating DM actions and player actions. The result of the search is an expected degree of interestingness of the different stories that will unfold depending on which DM action the DMM executes. Then, the DMM will select the DM action that maximizes such interestingness. Note that the *no-op* DM action (that represents executing no DM action) is always one of the DM actions considered.

In order to perform this search, the DMM uses a hybrid *montecarlo expectimax* algorithm, illustrated in Figure 4. The expectimax algorithm performs search by alternating two types of nodes in the search tree: *max* nodes and *exp* nodes. In the leaves of the search tree, the DMM will evaluate the interestingness of the story that unfolded to reach that leaf, and the interestingness value will be propagated up. In the *max* nodes, the interestingness is assigned as the maximum of the interestingness of the children nodes, and in the *exp* nodes, it is assigned as the average of the interestingness of the children nodes. Naturally, *exp* nodes will be nodes where the DMM is considering player actions, and *max* nodes correspond to nodes where the DMM is considering DM actions. The first node corresponds to the current game state, and is a *max* node. The next set of nodes is expanded by taking into account each possible DM action that can be performed in the current game state. This generates the first layer of nodes (*ply1*), where all the nodes are *exp* modes, as shown in Figure 4. The second set of nodes is generated by considering all the possible player actions, this generates the second layer (*ply2*), where all nodes are *max* nodes.

Since searching in such search tree has a very high cost, the DMM only performs exhaustive search for a predefined number of layers (set to 3 in our experiments). After that, the DMM performs *montecarlo search*. Montecarlo search estimates the interestingness value by expanding single branches at random, but

at a very hight depth (set to 20 in our experiments). The number of branches that the montecarlo part of the DMM expands is variable and depends on the amount of time left for search (as explained in the next section). At the end of each one of the montecarlo branches, the DMM uses the player model and the set of author specified guidelines to compute an interestingness value for that branch. Such interestingness is computed in the following way: first, the DMM uses the player model to predict the interestingness of each plotpoint visited in that branch, and each visible DM action executed in that branch, this gives a value *pmi*. Second, the DMM uses the author specified guidelines to compute a second interestingness value *asi*. The final interestingness is computed as $i = c \times pmi + (1 - c) \times asi$, where c is the confidence in the player model. As figure 4 shows, such interestingness values are propagated up the search tree, and finally the DMM will select the DM action that maximizes the interestingness. In the example shown in Figure 4, that corresponds to DM action dma_1, that achieves an interestingness value of 0.4.

Search-Based Drama Management on Real-Time Games Previous search-based drama management approaches have been implemented in turn-based games, where each time the player executes an action, the DM has time to perform a search and return a DM action. In our game the DMM is constantly running in parallel with the game as it evolves. In order to support this, the DMM constantly executes time bounded searches (2 seconds per search in our experiments). Each time a search is finished the DMM executes the proposed DMM action (if any). Each time the player (or any character in the game) executes an action, the DMM verifies that this action does not invalidate the search, and in case it is invalidated the search is restarted. In order to use the search time efficiently, the DMM uses an iterative deepening method that searches first only at depth 1, then depth 2, etc. As soon as the maximum level for exhaustive search is reached, then the DMM executes more and more montecarlo samplings, until the time is over.

Another challenge faced is that actions in our game take time, i.e. moving the character from a location to another takes some time (depending on the distance it has to travel), thus the DMM cannot just open a search tree where each node is a cycle of the game. In order to solve this, we incorporated the concept of *stable game state*. A stable game state is one where no character in the game is executing any action. The DMM always starts its search in a stable game state, and when an action is selected in a branch of the search, the game state is simulated until we reach another stable game state, from where the search can continue. Figure 4 illustrates the effect of this. On the left hand side we have the number of game cycles, and we can see that each level of the search tree does not correspond exactly with a particular game cycle.

The last challenge is related to the amount of actions the player can execute. Since in our game the set of actions is practically unbounded (the player can potentially command anything through natural language, restricted only by the limitations imposed by our NLU module and by the set of primitive actions

supported by our game engine). Notice for instance, that the player can type actions such as "go to the bar", "examine the ground", "say hello to the bum" or "ask the bum about his past"; just the set of possible communication actions to interact with other characters is unbounded. Since the DMM uses search based techniques (similar to those used in Chess playing programs), it needs a set of actions that the player can execute to perform the search. In our system, the DMM analyzes the set of plotpoints and the current game state to generate the set of actions that can affect the story. The rest of actions are considered as "no-effectors" by the DMM and are not considered in the search. Using this method, the author does not have to specify any list of possible actions (as in most previous DM approaches), the DMM can figure them out by itself.

3 Evaluation and Initial Analysis

We conducted our experiments in two batches. In the first, we invited five of the participants from [9] who had previously played an earlier turn-based text version of the game with an earlier version of our architecture. The second batch of ten participants had never played Anchorhead before.

The players in the first batch were interviewed at the end to get feedback on the usability improvements we made from the earlier version of our system namely shifting to a new graphical interface and employing a natural language based interaction modality. The first users acted as pilot subjects that helped us improve the graphical interface, NLP and DM modules. The suggested improvements from the first batch were implemented before the second batch of participants played the game. One of the improvement from their interaction was to improve the performance of the NLU by adding more lexical entries and rules. During their interaction, the users felt that the language based interface seemed like a more natural way of interaction and was much more interesting than the text based interface (where the user simply selected from a fixed list of possible actions presented to him as a choice list). However, the open ended nature of the NLU made it difficult for the users to figure out appropriate actions. In order to make it easier for users to find right set of actions, we improved the language interface with a help screen that provided them with example commands and their usage. We also added some more DM hints to help the next batch of users at the points where the first batch of users seemed to be generally lost.

After the improvements, we carried out a study with a second batch of participants to measure the effect of DM strategies on player's experience. For this batch of the study, we recruited 10 participants with a range of genders (2 females and 8 males). Each player was provided with an explanation on Anchorhead and asked to sign a consent form before starting the game. The player filled a background questionnaire to obtain information such as previous gaming experience or types of games they liked to play. During each episode, a researcher logged his observation of player actions and any unusual reactions. Each player played twice, once with DM active and another one without DM. On an average, the complete player interaction (both game playing episodes) lasted for about 45

minutes each. At the end of each episode, the player was asked to provide an interestingness value and an associated confidence value on a 5 point Likert scale for the overall game experience as well as the intermediate story events that were encountered during the interaction. At the end of both gaming episodes, participants were interviewed about their experience. The evaluation accounted for the order in which players played with and without DM by making half of the subject play first with DM and then without it, and the other half play first without DM and then with it. We transcribed the player responses from the interviews and observed players actions during the game episodes and analyzed them using a well-known qualitative analysis method, Grounded Theory [10] as used in [9]. Next we present results from qualitative analysis. We are planning to conduct a complete quantitative analysis in the future.

Concerning the interaction with the game, players found the natural language interface easy to use, although sometimes it was hard for them to figure out the appropriate actions due to the open ended NLU. Most players preferred the NLU interface to the choice-based interface (although a minority complained that it was annoying to type the commands). Further, having an open ended NLU interface made the players more engaged in the game as successfully taking an action was more rewarding (in the choice based interface it was obvious to figure out which actions to take next). Some players were frustrated as the open-ended NLU interface made them believe that they could interact freely with other characters, and in reality those character had very limited conversational topics (the characters in our game trigger responses or actions if the player talks to them about certain keywords or gives them certain objects).

Concerning the observable DM interventions some players felt that the DM was providing too many hints. Others liked the hints as they guided them in the game (when played without the DM some players were totally lost). Some other players felt that some hints were too blunt and told more than expected. (which points out that the DM should have several degrees of hints from more subtle to more blunt). Most players followed the hints and were able to finish the game when the DM was active, whereas without the DM most of them got stuck. Players felt a positive effect of the system's interventions on their interaction, when they saw that when playing without DM they ended up in an ending they didn't like, whereas playing with DM, the system (DM) prevented that ending from happening resulting in a more enjoyable experience. The DM hints also allowed certain players, who were stuck otherwise to finish the game.

4 Conclusions

In this paper we have presented an improved version of the drama management architecture initially presented in [9]. Specifically, our improvements include: real-time drama management, improved user modeling including preferences about DM actions, natural language interface for open ended user interaction (improving the DM to support this), and a graphical representation that is more

appealing to non-gamers. We performed a user evaluation and initial qualitative analysis with 15 users (5 users who had used the old system, and 10 new users).

The main conclusion we can draw from our experiments is that DM techniques are applicable to real-time games, and that DM generally improves player experience. Moreover, our user evaluation reinforced the fact that the DM should take into account player experience when deciding how to influence the story and also when deciding which hints to provide to the player.

As future work, we are in the process of conducting a quantitative analysis of our new DM architecture to measure how much the player experience improves with and without DM, and also how much it is due to the improved DM architecture with respect to the one reported in [9]. We are also in the process of developing a bigger game based on our architecture and plan to make it available on-line in order to have a larger number of players and perform further and better evaluations of the system.

References

1. A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1994.
2. J. Bates. Virtual reality art and entertainment. *The Journal of Teleoperators and Virtual Environments*, 2(1):133–138, 1992.
3. M. S. Gentry. Anchorhead. available online at http://www.wurb.com/if/game/17.html, 1998.
4. B. Magerko, J. Laird, M. Assanie, A. Kerfoot, and D. Stokes. AI characters and directors for interactive computer games. In *Proceedings of the 2004 Innovative Applications of Artificial Intelligence Confercence*, 2004.
5. M. Mehta and A. Corradini. Understanding spoken language of children interacting with an embodied conversational character. In *Proceedings of the Combined Workshop on Language-Enabled Educational Technology and Development and Evaluation of Robust Spoken Dialog Systems at ECAI 06*, pages 51–58, 2006.
6. M. Mehta and A. Corradini. Developing a conversational agent using ontologies. In *Proceedings of the International Conference on Human Computer Interaction*, pages 154–164, 2007.
7. M. Nelson, M. Mateas, D. Roberts, and C. Isbell. Declarative optimization-based drama management in interactive fiction. *IEEE Computer Graphics and Applications*, 26(3):33–41, 2006.
8. M. Sharma, M. Mehta, S. Ontañón, and A. Ram. Player modeling evaluation for interactive fiction. In *AIIDE 2007, Workshop on Optimizing Player Satisfaction*. AAAI Press, 2007.
9. M. Sharma, S. Ontañón, M. Mehta, and A. Ram. Drama management evaluation for interactive fiction games. In *AAAI-07 Spring Symposium on Intelligent Narrative Technologies*. AAAI Press, 2007.
10. A. Strauss and J. Corbin. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Sage, 1990.
11. P. Weyhrauch. *Guiding Interactive Drama*. PhD thesis, Carnagie Mellon University, 1997.