Integrating Story-Centric and Character-Centric Processes for Authoring Interactive Drama

Mei Si[1], Stacy C. Marsella[1] and Mark O. Riedl[2]

[1]Information Sciences Institute, University of Southern California
[2]School of Interactive Computing, College of Computing, Georgia Institute of Technology
{meisi, marsella}@isi.edu; riedl@cc.gatech.edu

Abstract

Computer aided interactive drama has been widely applied for entertainment and pedagogy. Most existing approaches for authoring interactive drama use either story-centric or character-centric processes. In this work, we present a new framework that integrates both types of processes to support authoring. This framework uses a multi-agent system to control virtual characters in a story. The characters' motivations are encoded as the agents' goals, and are configured based on well-structured story paths generated using a partial order planner. This framework allows the use of a planner that models the story at a more abstract level than the multi-agent system, and thus avoids the effort of building equivalent models of the story using both the planner and the multi-agent system. We explore the use of this new framework for authoring interactive dramas. Preliminary examples of application are presented.

Introduction

Computer aided interactive drama allows the user to actively participate in a story, by playing a role or applying directorial control over the characters in the virtual world. The user's choices affect the unfolding of the story. Compared to traditional drama, the integration of narrative and interactivity enables interactive drama to create richer and more engaging experience. Therefore, it has been widely applied for providing both pedagogy (e.g. Louchart & Aylett 2004; Traum et al., 2005; Si et al., 2005; Riedl et al., 2008) and entertainment (e.g. Cavazza et al., 2001; Mateas & Stern, 2003; Szilas, 2003; Braun, 2003; Young et al., 2004; Magerko, 2005).

One of the central challenges faced in the design of interactive drama is how to reduce authoring effort resulting from the merge of interactivity and narrative. Unlike traditional drama, in which only a single story line is presented to the user, interactive drama allows the user to interact with the virtual characters. Authoring enough contingencies to create a richly interactive environment for an engaging experience is often intractable to human authors (Riedl & Young, 2006).

To address this challenge, various authoring frameworks have been designed for facilitating human author's creation of interactive dramas. Many of these authoring frameworks adopt approaches inspired by theories of what makes a good story. In *Poetics*, Aristotle argued that character was subsidiary to action. A more contemporary view on character and action, as espoused by Lajos Egri (Egri, 1949), suggests that plot unfolds based on the characters, that characters can essentially "plot their own story". Corresponding to the above theories, *story-centric* processes (e.g. Mateas & Stern, 2003; Szilas, 2003; Braun, 2003; Young et al., 2004; Magerko, 2005; Riedl et al., 2008) for interactive drama focus on the structure of the overall story in terms of plot arc, and aim at providing automated approaches for arranging events to happen during the interaction to form a well-structured story. *Character-centric* processes (e.g. Cavazza et al., 2001; Louchart & Aylett, 2004; Traum et al., 2005; Si et al., 2005), on the other hand, emphasize the development of individually plausible, autonomously motivated characters that the user can interact with.

In this work, we present a new framework that integrates story-centric and character-centric processes for authoring interactive dramas. This new framework integrates a partial order planner (POP), which has often been used in story-centric processes, with the Thespian system (Si et al., 2005), which mainly uses a character-centric approach. The details of the integration are provided in this paper, followed by preliminary examples of human author interacting with the framework to author an interactive drama.

Example Domain

The example domain of this work is a Grimms' fairy tale, "Little Red Riding Hood''. The story starts as Little Red Riding Hood (Red) and the wolf meet each other on the outskirt of a wood while Red is on her way to Granny's house. The wolf has a mind to eat Red, but it dares not because there are some wood-cutters close by. The wolf will eat Red at other locations where nobody is around. Moreover, if the wolf hears about Granny from Red, it will even go eat her. Meanwhile, the hunter is searching the wood for the wolf. Once the wolf is killed, people who were eaten by it can escape.

Overview of the Framework

This new framework integrates a POP planer with Thespian system for facilitating the author in creating interactive dramas.

POP planners have often been used in story-centric authoring processes (Young et al. 2004; Riedl et al. 2008) because they can automatically generate sequences of the characters' actions – plans to reach story goals and at the same time ensure plausible causal relationship among events in the plan. However, such plans do not provide the author insight about the characters' motivations, and therefore cannot avoid creating inconsistent character motivations during the interaction.

Thespian (Si et al., 2005) on the other hand mainly adopts a character-centric approach and can ensure consistent characters' motivations during the interaction. It uses decision-theoretic goal-driven agents to control virtual characters. The characters' motivations are encoded as the agents' goals. Thespian provides an automated fitting procedure which can tune virtual characters' motivations to a set of story paths (sequences of the characters' actions). The resulting virtual characters will recreate their roles when the user's actions are the same as specified in the story paths. When the user deviates from the story paths, the characters will respond to the user using the motivations "learned'' from the story path fitting process. However, deviation from the author designed paths risks the interaction being a not well-structured story. To account for this, the author often needs to design multiple story paths for configuring virtual characters.

In this new framework, we use a planner to partially automate the story path designing process. The author works with the planner to construct story outlines – represented as plans – and the plans are then used to provide guidelines for configuring Thespian agents, which are later used for interacting with the users. To avoid the effort of constructing an equivalent model of the story in the planner as that in Thespian, this framework allows the use of a planner that models only major events (plot points) of the story. The framework provides a procedure that can tune Thespian agents' motivations to plot level plans.

Figure 1 lays out the overall authoring process of this framework. The authoring process starts with the author using the planner to produce one or more plot level skeletons (plans) of the story. The author does so by providing specifications for what they would like to see in the plot. For example, the author might specify that Granny is eaten by the wolf. The planner responds by generating a plan such as: Red walks to Granny's cottage, Red tells the wolf about Granny and then the wolf eats Granny.

Next the system works with the author to elaborate the plot level plan into a full story path. Since the Thespian agents may model the story with more detailed interactions, their motivations may not be directly fitted to the plot level plan, e.g. there is no feasible motivation for Red to tell the wolf about Granny as soon as they meet. In this case, the system first tries to fill in moment-to-moment interactions between the two plot points, such as a small talk between Red and the wolf to build rapport. If the system cannot find appropriate interactions or the author does not like the suggested interactions, the system may take initiative in authoring and propose new actions to be modeled in Thespian agents, e.g. an action that can make Red believe that the wolf is trust worthy. The system can only suggest the effects of the actions, and the author needs to respond by inventing the actions, e.g. a wood-cutter tells Red that the wolf is trustable.

Finally, Thespian agents "learn" their motivations from the full story path and any other paths designed previously by the author or the system, and interact with users.

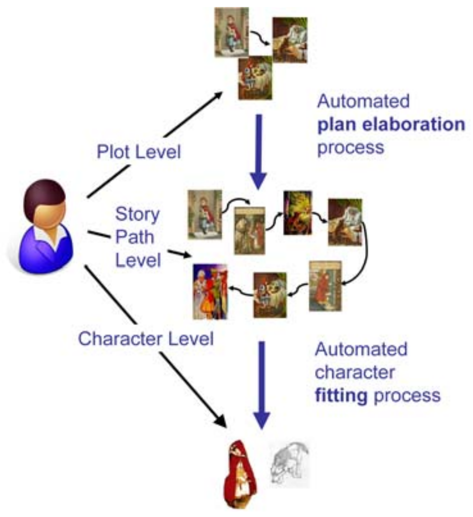

Figure 1. Overview of the New Framework

Implementation

In this section, we introduce the Thespian system, and its integration with a POP planner. A standard POP planner is assumed to be used, so no detail about the planner is included in this paper.

Thespian

Thespian is a multi-agent system for authoring and controlling virtual characters in interactive dramas. It is built upon PsychSim (Marsella et al., 2004), a multi-agent system for social simulation. In this section, we introduce components in Thespian that are relevant to this work.

Thespian Agent

Thespian's basic architecture uses POMDP (Smallwood & Sondik 1973) based agents to control each character, with the character's motivations encoded as agent goals. Each agent is defined by its state, action, action dynamics, goals, policies, and beliefs about self and others.

An agent's **state** is defined by a set of state features, such as its name and location. The agent's **actions** are the same as the actions of the character in the story world, such as talking to other characters and moving around. The agent's **action dynamics** define how its state is affected by events (actions of characters) happen in the story. The **goals** of an agent are expressed as a reward function over the various state features the agent seeks to maximize or minimize. For example, the wolf character may have goals of satisfying its hunger and keeping itself alive, with the latter one having much higher importance. Agents have **recursive beliefs** about self and others, e.g. the wolf's belief about the hunter's belief about itself. This forms a model of theory of mind. This model enables Thespian agents to reason about other characters' reactions when planning on their own behaviors. Currently, all agents use a bounded **lookahead policy** to decide their actions -- it projects limited steps into the future to evaluate the utility of each action option. The agent considers not just the immediate effect of an action, but also the expected responses of other characters and, in turn, its reaction to those responses and so on. The agent evaluates the overall utility with respect to its goals and then chooses the action that has the highest expected utility. For example, the wolf chooses to talk to Red instead of eating her because it foresees the outcome of being killed by the wood-cutter after eating Red, and the goal of being alive is far more important for it than keeping itself from being hungry.

"Fitting" Procedure and "Suggest" Procedure

"Fitting" and "suggest" are two procedures provided by Thespian to help the author design virtual characters which will behave in a certain (desired) way.

The fitting procedure enables an author to configure virtual characters' goals by creating alternative paths of the story. It configures each virtual character separately. For each virtual character, it judges if consistent motivations can be inferred from the story paths – whether there is a set of goal weights that can motivate the agent to behave as specified in the story paths. If the answer is yes, it sets the agent's goal weights to that solution, and otherwise informs the author of the failure. See (Pynadath & Marsella, 2004; Si et al., 2005) for details.

"Suggest" is a procedure provided by PsychSim. This procedure can suggest belief changes to an agent (without affecting the agent's goals) so that it will prefer the author's desired choice of action over its original choice. For example, to make the wolf not eating Granny, the "suggest" procedure may give the following solution: make the wolf believe that it is not hungry (instead of being hungry). The author then needs to arrange an event, which can create the belief change, to happen in the story before the wolf needs to make its decision about eating Granny. Currently this procedure assumes that the agent can expect others to always react to its actions the same way regardless of its belief changes, e.g. no matter how the wolf's belief changes, it always expects to be killed by the wood-cutter if it eats Red (this expectation is formed when the agent did its lookahead reasoning with its original beliefs). This is not always appropriate. For example, if the wolf changes its belief to that the wood-cutter is not close by it does not need to worry about being killed. In section Implementation, Algorithm 5 extends the "suggest" procedure to consider this situation.

Elaborate Plot Level Plan into Full Story Path

In this framework, we replace the hand authored story paths in Thespian's authoring procedure with story paths generated by a planner, and use plans to provide guidance for virtual characters' behaviors. It is straightforward to tune Thespian agents' motivations to plans generated at the same detail level as that used by Thespian model; the fitting procedure can be directly called. When configuring Thespian agents to behave according to plot level plans, each plan is first elaborated into a full story path. The Thespian agents are then fitted to these story paths and any additional ones designed by the author or the system. This section presents the algorithms for elaborating a plot level plan into a full story path.

Identify "Gaps" in a Plan

When a plan is passed to Thespian, no special tag is needed to indicate whether it is a plot level plan. Instead, the system uses automated procedures to find out if there are moment-to-moment interactions missing from the plan. In other words, whether there is a "gap" in the plan. If the answer is yes, the next step is to determine what actions should be inserted into the plan and where they should be inserted. This information is returned to the author as feedback. These steps may need to be repeated multiple times until all the gaps in the plan are filled. Algorithm 1 is used for locating the first gap in a plan.

Algorithm 1 Identify_Gap (*plan*)

```
1. If fit (plan [1: len(plan)]):
2.     Return -1
3. Else:
4.     For i = 1: len(plan):
5.         If ~fit (plan[1:i]):
6.             Break
7. Return i
```

Algorithm 2 Fit (*seq*)

```
1. For character in story:
2.     If fit_sequence( character,seq) == False:
3.         Return False
4. Return True
```

Algorithm 1 first tries to treat the entire plan as a regular story path. If virtual characters can be successfully fitted to the plan, it is suggested that the planner models the story at the same level as Thespian, and no further actions need to be taken. Otherwise, this algorithm progresses stepwise to find the first gap in the plan. Starting with *i* equals to 1, it fits the virtual characters to the first *i* actions in the plan. If it succeeds, it fits the virtual characters to the first $i+1$ actions in the plan. When fitting fails, we know that there is a gap around the i^{th} action and the $i+1^{th}$ action in the plan, and additional actions need to be inserted either between these two actions or before the i^{th} action. For example, the plan passed to Thespian may be: Red walks to Granny's cottage, Red tells the wolf about Granny … When fitting Thespian agents to this plan, the agent that controls Red can be fitted to the first action, but not the first two actions. Therefore, a gap is found. The ***fit*** function used in Algorithm 1 is defined in Algorithm 2. It calls the fitting procedure in Thespian and fits all the characters in the story to a sequence of actions, and only returns true if all the characters can be successfully fitted.

Fill Gap

When a gap occurs in the plan and needs to be filled with moment-to-moment interactions, usually the solution is not limited to a unique one. Further, different ways of filling the gap shapes the resulting story path and the Thespian agents, which will be used to interact with the user, differently. For example, the simplest way to fill the gap, that Red will not tell the wolf information about Granny when they first meet, is to add small talk between Red and the wolf. The small talk will gradually build rapport between the two characters. Alternatively, more complex stories can be made, such as a wood-cutter happens to pass by and he convinces Red that the wolf is a good friend.

In this new framework, the author is given freedom to specify which actions to use for filling a gap, and their priorities. Upon the identification of a gap, the system automatically divides virtual characters' actions into three categories based on how much the action can potentially change the story when used for filling the gap. The first category contains small talk actions that involve only the characters related to gap. The characters related to the gap are those who act right before or after the gap, e.g. Red and the wolf in the previous example. The second category contains these characters' other actions, such as talking to other characters and moving around. The third category contains other characters' actions, such as the wood-cutter's actions. Adding small talk actions between related characters affect the story least, because small talk is most likely to be omitted when modeling a story at an abstract level; and it is assumed that unrelated characters' actions can potentially change the story most dramatically.

In addition to what actions can be used for elaborating the plot level plan, the location of the inserted interactions, e.g. within or right before the gap vs. several steps before the gap, and the length of the inserted interactions also affect the difference between the resulting story path and the original plot level plan. Algorithm 3 & 4 give the pseudo code for filling gaps based on these parameters.

As shown in Algorithm 3, the plot level plan is first cut into three parts. *Path0* contains the sequence of actions before the gap. *Islands* contain the two actions around the gap and the *n* actions that immediately precede the gap. The basic idea is to replace *islands* with detailed moment-to-moment interactions; all the actions in *islands* need to be included in the final story path with their original order kept. This way, moment-to-moment interactions are inserted between the events in *islands*. Finally, the third part of the plan is the sequence of actions that happen after the gap. Currently they do not affect how the gap is filled.

Algorithm 3 Fill_Gap (*plan, i, allActionSets, n=0, maxLength*)

```
1:  # i: location of the gap in plan
2:  # allActionSets: sets of actions to be used. The sets are ordered in
            descending priorities.
3:  # n: the starting location for filling
4:  # maxLength: maximum length of interaction allowed

5:  res ← False
6:  path0 ← plan[0, i- n]
7:  islands ← plan[i- n, i+2]
8:  actionSet ← allActionSets [0]

9:  res ← replace_islands (path0, islands, actionSet, maxLength )
10: If res == False:
11:     For newActions in allActionSets [1:]:
12:         actionSet ← actionSet + newActions
13:         res ← replace_islands (path0, islands, actionSet,
        maxLength )
14:         If res == True:
15:             Break
16: Return res
```

Algorithm 4 Replace_Islands (*path0, islands, actionSet, maxLength*)

```
1:  res ← False
2:  For action in actionSet:
3:      path ← copy(path0)
4:      path ← path + action
5:      If checkOrder(path, islands):
6:          If checkComplete(path, islands):
7:              res ← fit (path)
8:              If res == True:
9:                  Return True
10:         Else:
11:             maxLength ← maxLength -1
12:             If maxLength >= 0:
13:                 res ← replace_islands (path, islands,
                    actionSet, maxLength)
14:                 If res == True:
15:                     Return res
16: Return res

17: # checkOrder(path, islands): returns if the order of actions in islands
        is retained in path. Only applies to those actions appear in path.
18: # checkComplete(path, islands): returns if each action in islands is
        included in path
```

The author can indicate the sets of actions to be considered for filling the gap with priorities using the *allActionSets* parameter. Initially only the set with highest priority is used (line 8 in Algorithm 3). If it fails to fill the gap, the set of actions with next highest priority will be added for consideration (line 11-12 in Algorithm 3).

Algorithm 4 illustrates how actions are taken from the allowed action sets (*actionSets*) and combined together to replace *islands* in the plan. As actions are appended one by one to the end of *Path0* (line 4 in Algorithm 4), the function keeps on checking if the story path satisfies the basic requirements for replacement – all actions in *islands* are included in the story path with their original order (line 5-6 in Algorithm 4). If the story path passes this checking, the function will try to fit virtual characters to the story path (line 7 in Algorithm 4). If fitting succeeds, the gap is successfully filled. In the worst case, this recursive function will try all the combinations of actions from *actionSets* where the length of the sequence is equal to or less than *maxLength*.

Algorithm 5 Suggest_Belief_Changes (*plan, i*)

```
1:  # i: location of the gap in plan
2:  actor ← the agent who performs the action plan [i+1]
3:  options ← []

4:  simulate the story until the i-th step of the plan
5:  options ← options + Suggest_Pick (actor, plan [i+1])

6:  For other in story:
7:      For otherAct in other.actionOptions():
8:          If lookahead (actor, other, otherAct) == plan [i+1]:
9:              simulate actor performs the action plan [i+1]
10:             options ← options + Suggest_Pick (other, otherAct)
11: Return options

12: # Suggest_Pick (actor, action): returns necessary changes to the
    actor's beliefs so that the actor will pick action over all other choices
13: # lookahead (actor, other, otherAct): returns actor's choice after
    lookahead with the expectation of other's response being otherAct
```

It is possible that Algorithm 3 fails to fill the gap. In this case, just using actions that have already been modeled in Thespian is not enough to recreate the story laid out by the planner. The system can take initiative in authoring by suggesting new actions to be included in Thespian's model for filling the gap. Algorithm 5 illustrates this process.

Algorithm 5 extends the "suggest" procedure in Thespian / PsychSim. It first simulates the story until the ith action of the plan, where the gap happens. *Actor* is the agent who will act next. At line 5, Algorithm 5 identifies belief changes that if happen *actor* will select the $i+1$th action in the plan as its next step. Further, Algorithm 5 considers the case that *actor*'s choice is affected by its expectations of other characters' responses. For each potential response from other characters (line 6-7 in Algorithm 5), if *actor* will choose the desired action when expecting that response (line 8 in Algorithm 5), Algorithm 5 proceeds and finds out necessary belief changes if any that will lead the character to make the response (line 10 in Algorithm 5). The pseudo code in Algorithm 5 illustrates considering the *actor*'s anticipation of other characters' responses using one step lookahead. It is straightforward to extend the pseudo code for more steps of lookahead.

After belief changes are proposed by the system, the author needs to respond by creating corresponding actions that can result in the belief changes. This process often generates interesting and creative ideas. An example is given in the next section.

Authoring Examples

In this section, two examples of using this new framework to author the Red Riding Hood story are provided[1]. Both examples start with the plot level plan and demonstrate how the framework interacts with the author to configure Thespian agents.

Example 1

This example demonstrates how small talk actions can be used to turn a plot level plan into a complete story path. The planner produced the following plan.

1. The wolf comes to Red (on the road).
2. Wolf: where are you going?
3. Red: I am going to Granny's house to give her this cake.
4. …

The system applied Algorithm 2 and found a gap between the wolf's enquiry and Red's reply. Next, the system tried to fill the gap using only small talk actions and succeeded. The following story path was generated and returned to the author as feedback. The Thespian agents' motivations were also tuned to this story path.

1. The wolf comes to Red (on the road).
2. Wolf: hello!
3. Red: hello!
4. Wolf: how are you?
5. Red: I am doing well.
6. Wolf: where are you going?
7. Red: I am going to Granny's house to give her this cake.

The author approved the generated path and the system proceeded to identify the next gap in the plan.

Example 2

This example demonstrates how belief changes in characters are suggested for linking plot points in the plan; and how these belief changes can be turned into novel actions of Thespian agents.

The plot level plan indicates that the following scenario should happen after the wolf eats Granny in the cottage.

1. Red comes to the door.
2. Red enters the cottage.
3. The wolf eats Red.

The system applied Algorithm 2 and found there is a gap between "Red enters the cottage" and the previous step if Red knows the wolf is inside. The system then tried to fill the gap using all the modeled actions and found only one solution: the wolf leaves the cottage before Red enters. However, the author disapproved this solution because they foresee that it will make the later action – the wolf eats Red (in the cottage) – impossible to happen. Next, the system applied Algorithm 5 and got the following results:

- Red believes that it is Granny who is inside
- Red believes that the wolf is somewhere else
- Red believe that the wolf is full
- Red believes that the wolf thinks she is dead (the logic behind this is that the wolf will not eat a dead person).

[1] For better readability, the examples are given using the actual sentences in the story instead of speech acts, which are used by the system for reasoning internally.

Each of these belief changes can make Red expect to not be eaten after entering the cottage and therefore enters the door. Based on these suggestions, the author designed several new actions, including “the wolf disguises itself as Granny”, “the wolf pretends it is leaving” and “the wolf eats food in Granny’s kitchen”. These actions were added to Thespian agents’ models and the system tried to fill the gap again. The following final story path was generated:

1. The wolf ***disguises*** itself as Granny.
2. Red comes to the door.
3. Red enters the cottage.
4. The wolf eats Red.

Discussion and Future Work

In the examples shown above, the gaps are filled independently, i.e. we did not consider how the filling of a gap affects the difficulty of filling latter gaps in the plan. For future work, we plan on enhancing the plan elaboration process by taking dependencies among gaps into considerations.

The framework has been fully implemented, but we have only used it to author one interactive drama. In this story, the main characters (Red and the wolf) have around 10 different action choices and other characters have 3-4 action choices. In the future, we are interested to exam how this framework works in a more complex domain.

Conclusion

In this paper, we present a framework that integrates story-centric and character-centric processes for facilitating the human author in creating interactive drama. This framework is implemented by using story paths generated by a POP planner for configuring Thespian’s goal-based agents. The automation in story path generation makes it easier to provide multiple well-structured story paths for fitting Thespian agents; and with more training examples, we can expect a better chance of the user experiencing a well-structured story when interacting with the virtual characters. This framework allows the use of a planner that models the story at a more abstract level than Thespian. This saves the author effort of building an equivalent model of the story in the planner as that in Thespian, and thus enables the author to quickly sketch the interactive experience (using the planner).

References

Braun, N. 2003. Storytelling in collaborative augmented reality environments. In *Proceedings of the 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision.*

Cavazza, M., Charles, F., and Mead, S.J. 2001. Agents’ interaction in virtual storytelling. In *Proceedings of the International Workshop on Intelligent Virtual Agents.*

Egri, L. 1949. The Art of Dramatic Writing. *Simon & Schuster: New York.*

Louchart, S., and Aylett, R. 2004. The emergent narrative theoretical investigation. In *Proceedings of the 2004 Conference on Narrative and Interactive Learning Environments.*

Magerko, B. 2005. Story representation and interactive drama. In *Proceedings of the 1st Conference on AI and Interactive Digital Entertainment.*

Marsella, S.C., Pynadath, D.V., and Read, S.J. 2004. PsychSim: Agent-based modeling of social interactions and influence. In *Proceedings of the International Conference on Cognitive Modeling, 243–248.*

Mateas, M., and Stern, A. 2003. Integrating plot, character and natural language processing in the interactive drama Façade. In *Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment, Darmstadt Germany.*

Pynadath, D.V. and Marsella, S.C. 2004. Fitting and Compilation of Multiagent Models through Piecewise Linear Functions. In *AAMAS.*

Riedl, M.O. and Young, R.M. 2006. From Linear Story Generation to Branching Story Graphs. *IEEE Computer Graphics and Applications*, 26(3).

Riedl, M.O., Stern, A., Dini, D., and Alderman, J. 2008. Dynamic Experience Management in Virtual Worlds for Entertainment, Education, and Training. *International Transactions on Systems Science and Applications, Special Issue on Agent Based Systems for Human Learning*, 4(2).

Si, M., Marsella, S.C., and Pynadath, D.V. 2005. THESPIAN: An Architecture for Interactive Pedagogical Drama. In *AIED.*

Smallwood, R.D., and Sondik, E.J. 1973. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, *21:1071–1088.*

Szilas, N. 2003. IDtension: a narrative engine for interactive drama. In *Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment, Darmstadt Germany.*

Traum, D.R., Swartout, W., Marsella, S.C., and Gratch, J. 2005. Fight, flight, or negotiate: Believable strategies for conversing under crisis. In: *IVA.*

Young, R.M., Riedl, M.O., Branly, M., Jhala, A., Martin, R.J., and Saretto, C.J. 2004. An Architecture for Integrating Plan-Based Behavior Generation with Interactive Game Environments. *Journal of Game Development, 1, 51-70.*