Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment

Robust and Authorable Multiplayer Storytelling Experiences

Mark Riedl, Boyang Li, Hua Ai, and **Ashwin Ram**
School of Interactive Computing
Georgia Institute of Technology
Atlanta, Georgia 30308
{riedl, boyangli, hua.ai, ashwin}@cc.gatech.edu

Abstract

Interactive narrative systems attempt to tell stories to players capable of changing the direction and/or outcome of the story. Despite the growing importance of multiplayer social experiences in games, little research has focused on multiplayer interactive narrative experiences. We performed a preliminary study to determine how human directors design and execute multiplayer interactive story experiences in online and real world environments. Based on our observations, we developed the Multiplayer Storytelling Engine that manages a story world at the individual and group levels. Our flexible story representation enables human authors to naturally model multiplayer narrative experiences. An intelligent execution algorithm detects when the author's story representation fails to account for player behaviors and automatically generates a branch to restore the story to the authors' original intent, thus balancing authorability against robust multiplayer execution.

Introduction

An *interactive narrative system* attempts to engage players in a story under the conditions that the players' behaviors and actions can directly impact the direction and/or outcome of the story. That is, interactive narratives balance meaningful choices by players against the coherence and quality of the resultant narrative experience as dictated by a human author. A *drama manager* (Laurel 1986) is one common approach to balance the competing requirements of player agency and authorial intent. A drama manager is an omniscient agent that observes the story world and directs non-player characters (NPCs) in order to bring about the human author's intent in the face of uncertainty resultant from a relatively under-constrained interactive player.

A vast majority of interactive narrative work has focused on single-player experiences. However, there are a number of multiplayer environments that can benefit from what we know about interactive narrative. For example, an *Alternate Reality Game* (ARG) is an interactive narrative that uses the real world as a platform, layering a fictional world over the real world. ARGs place high demands on human game masters and, as such, are limited in scale until games can be managed autonomously (Barve et al. 2010). Multiplayer training simulations in online worlds and the real world can also benefit from experience management (Raybourn et al. 2005; Niehaus and Riedl 2009). In addition to multiple participants in the interactive narrative, it is also desirable for each participant to experience different narrative arcs within a larger narrative structure and to be manipulated into cooperative or adversarial situations. That is, a drama manager must maximize the value of each individual player's experience while also maximizing the overall narrative experience.

In the work reported here, we asked experts on multiplayer alternate reality games to create and manually run two interactive multiplayer alternate reality experiences using human confederates in the place of non-player characters. Based on our observations of the creation and execution of the story, we derived the following set of properties that multiplayer interactive narratives should possess:

- *Multiplayer differentiability.* Multiple players can participate in the same game and have potentially distinct narrative experiences.
- *Authorability.* The human author can easily articulate how different players experiences should unfold.
- *Robustness.* The system should be able to handle situations where the authored story does not provide guidance on how to alter the narrative experience in response to unanticipated player behaviors.

One of the core challenges in the development of an interactive narrative is determining how much the drama manager must rely on pre-authored story content from the human author. On one end of the spectrum, we have Choose-Your-Own-Adventure novels, in which a tree of plot points is completely pre-authored. On the other end of the spectrum, a *generative drama manager* (Riedl et al. 2008; Young et al. 2004) constructs the story from scratch based in response to the behaviors of the player. Other approaches such as that used in the successful FAÇADE (Mateas and Stern 2003) interactive drama attempt to strike a balance. We observe that as the story world includes more players and affords greater expression to the players, the ability of the human author to predict and control player behavior—especially when multiple players can interact with each other outside the control of the experience manager—decreases, requiring greater gen-

erative power. We cast this as a tradeoff between authorability and robustness in the face of multiplayer differentiability.

We present an authorable narrative representation and drama management execution process that lends robustness to story progression in circumstances where players perform behaviors unanticipated by the human author. Our representation is a specialization of Colored Petri Nets (Jensen 1996) that models an interactive narrative as a process in which multiple players flow from scene to scene. Our execution algorithm monitors for situations in which the Petri Net fails to account for player actions and automatically constructs new story branches to restore the Petri Net. In the remainder of the paper we situate our approach in the context of related work, briefly describe the preliminary study with expert alternate reality game directors that informed our approach, and describe the representation and execution algorithm for the Multiplayer Storytelling Engine (MUSE).

Related Work

Drama management, an approach to interactive narrative, employs an intelligent and omniscient agent able to monitor the virtual world and intervene in the world to ensure coherent story progression while simultaneously providing the player with a large degree of freedom to act. These interventions are typically instructions to non-player characters (NPCs) to engage with the player in a certain way, but may also physically change the world. Riedl et al. (2003) categorize player actions in interactive narratives based on how they affect the story the drama manager intends to tell:

- *Contingent*: An action performed in line with the author's expectations, thus progressing the story.
- *Consistent*: An action that changes the world in a way that neither assists nor hinders progression of the story.
- *Exception*: An action that alters the world in a way that hinders story progression by making a future player or NPC actions illegal or nonsensical.

All drama managers are dependent to some extent on knowledge provided by the human designer, which may include large fragments of story, primitive actions to be assembled at run-time, and/or rules on how to handle exceptions. The body of drama management work can be described in terms of a spectrum of knowledge requirements running from *author-intensive* to *generative*, with associated pros and cons. Author-intensive approaches have considerable authorial burden for any significantly large story world (Bruckman 1990) but can be highly effective. Generative drama managers can respond to a greater range of player behaviors but require story generation capabilities that are not yet at the level of human storytelling competency. Space precludes an in-depth review of drama management techniques; we note the most relevant work.

Balas et al. (2008) use hierarchical Petri Nets so that authors can completely manually construct and execute complex, branching plot structures. As with all heavily authored approaches, success is dependent on the authors' ability to account for all possible exceptions at every point in the narrative. A common approach that strikes a middle-ground is to author *plot graphs* that specify ordering dependencies between possible plot points coupled with some form of search to construct a policy for how and when a drama manager should intervene (Weyhrauch 1997; Nelson et al. 2006; Sharma et al. 2010). The FAÇADE interactive drama (Mateas and Stern 2003) also uses numerous hand-authored plot points and character behavior decompositions coupled with intelligent plot point and behavior selection. We note that the greater the extent that a drama manager relies on human-authored content, the greater the conflation of robustness with authorability, as the system is only as robust in so far that the author is able to anticipate all exceptions.

Generative drama management, in contrast, utilizes a narrative generation system to construct novel story sequences whenever the player performs an exception (Riedl, Saretto, and Young 2003; Young et al. 2004; Barber and Kudenko 2007; Riedl et al. 2008; Porteous and Cavazza 2009). Many generative drama managers represent story as partial-order plans and use a planner to generate and/or adapt the story plan when necessary. Partial-order plans, in the case of storytelling systems, are graphs of events (or scenes) where arcs capture causal and temporal relations. Partial-order plans have been found to be good representations of narrative for interactive systems for the following reasons: (a) partial ordering supports the parallelism of action that can occur in stories; (b) the causal relations tracked by partial-order plans are useful in identifying and handling exceptions; (c) causal structures may correlate to certain aspects of aesthetics (Trabasso and van den Broek 1985); and (d) *plan refinement* algorithms can easily modify the existing story structure with new scenes, thus implementing a form of continuous planning in response to player exceptions.

The IN-TALE (Riedl et al. 2008) system is a generative drama manager that starts from a pre-authored, non-branching narrative plan and automatically generates branches to handle exceptions. Our work is similar to IN-TALE in that we also start from an authored narrative specification; however, we start from a pre-authored structure that captures branching and accounts for different narrative arcs for different players, thus accounting for authorability, robustness, and multiplayer differentiability.

Historically, drama management has focused on single-player experiences. Winegarden and Young (2006) extended the MIMESIS generative drama manager to account for multiple, distinct stories executing in the same environment. Their work only considered load balancing between "story servers" that may occasionally interfere with each other, but didn't consider the problem of managing multiple players in the same narrative experience. Thus a novel contribution of our work is an approach that specifically caters to multiplayer environments with the goal of using story to create rich social interactions.

Preliminary Study

We conducted a preliminary, informal study to explore the problem of multiplayer interactive narrative. We recruited a director/screenwriter—an expert at crafting multiplayer alternate reality games—to write two multiplayer stories that would engage a half dozen people periodically over several

hours. He was encouraged to use whatever visual representation he felt appropriate to capture his design.

The director created the stories as graphs, where each node represented a scene, a general description of the type of activity that those present in a particular place would engage in such as meeting a character or stealing an item. Scenes were meant to be able to unfold in many different ways and have many different outcomes. Edges between nodes acted as filters selecting which players would progress to which subsequent scenes. Thus, players had the potential to experience different sub-stories. We also note that the director used early scenes to "matchmake"—to cluster players into groups—using lots of conditional edges and optional scenes, and used later scenes to create periods of competition and cooperation between groups of players.

As in the study by Kelso, Weyhrauch, and Bates (1993), participants were recruited to play the game and confederate actors assumed the roles of NPCs. The human director, with the help of assistants, manually gave instructions to NPCs based on the story graph. Each story was run five times over the period of a week with different participants each time. Space prevents further details in this paper; however, we made the following relevant observations:

1. Despite the best efforts of the director to predict different ways scenes could occur, exceptions happened.
2. The director responded to exceptions by creating new scenes on the fly that, when possible, restored the players to the pre-authored scenes by the shortest reasonable means.
3. In circumstances where the director deemed it too difficult to restore players to the pre-existing scenes, the director authored new chains of scenes that skipped pre-authored scenes and rejoin the story at a later point in the graph or to bring the story to an early resolution.

Based on our observations of the human director during the preliminary study, we developed the Multiplayer Storytelling Engine described in the next section for managing multiplayer differentiable experiences in online or real world story environments.

The Multiplayer Storytelling Engine

The Multiplayer Storytelling Engine (MUSE) combines human-authored story content and a robust generative execution process that handles exceptions that are unanticipated by the human author. The representation for story content must support multiplayer scaling and be easily authorable by designers without requiring programming or manipulation of complicated AI structures such as plans. To achieve robust execution, the representation must support the ability for an AI system to recognize and autonomously respond to exceptions by dynamically generating a new branch to the story structure. Thus, the approach behind MUSE is a collaboration between offline human storyteller and an online AI system.

Story Representation

We represent multiplayer interactive narratives as a variation of Colored Petri Nets (CPNs) (Jensen 1996), where *places* are scenes, *tokens* are players, and *transitions* encode the "ideal" progression of individual players through the scenes. We chose the CPN representation because of ease of authoring branching story structures (Balas et al. 2008) as well as similarity to the graphical construct devised by the director in the preliminary study. The representation of players as tokens emphasizes a player-centric philosophy to storytelling, allowing the author to determine how individual players flow through a story structure. This is more expressive than a traditional branching story in that it can capture both branching and parallelism; different players can be engaged in different scenes at the same time.

While Petri Nets can represent many different types of processes, a Petri Net assumes that all faults in a process are handled with transitions. The preliminary study demonstrated that even experts cannot make fault-free narrative structures in worlds that allow a high degree of player expression and agency. Consequently, we modify the CPN representation with information that allows an intelligent execution engine to dynamically modify the network structure to handle unanticipated exceptions.

We represent a scene as a tuple $\langle m, A, B, P, E \rangle$ where:

- m is the minimum number of tokens required to execute;
- A is a list of NPC agents that will participate in the scene;
- B is the body, the set of instructions to team surrogates;
- P is a set of predicates describing conditions in the world that must be true prior to execution; and
- E is a set of predicates that will become true if the scene executes successfully.

Unlike traditional CPNs, scenes do not execute until a minimum number of tokens are located at the scene and the precondition predicates are satisfied by the current world state.

Transitions dictate storytelling logic by determining which scene each player token should experience next based on what happened in the prior scene. That is, instead of implementing scene applicability constraints (which are handled by the scenes themselves), transitions act as a filter on tokens. Transitions are thus constrained to be of the form $\langle s_1, s_2, P, C \rangle$ where s_1 and s_2 are the originating and terminal scenes respectively, P is a list of players (or "any") in scene s_1 to test, and C is a (possibly empty) list of predicates referencing players in P that will be tested against the current state of the world. For example, a transition may select players that have, in the course of a scene, agreed to help a particular NPC (e.g., the condition `agree(`*player*`?, `*npc*`?)` is true), while another will select the opposite.

Figure 1 shows an example Petri Net for a MUSE story; for clarity, it is simpler than that from the preliminary study. There are eight players, who are tasked with solving a puzzle in the first scene. The transitions on the first scene sort the players based on those who solve the puzzle and those who do not. The first half of the network engages the players in a set of tasks that matchmake the players onto teams aligning with a pirate, a witch, and a navy captain. The remainder of the scenes bring the teams together in various cooperative and competitive configurations. Note the use of transitions to separate teams after they come together for joint scenes.

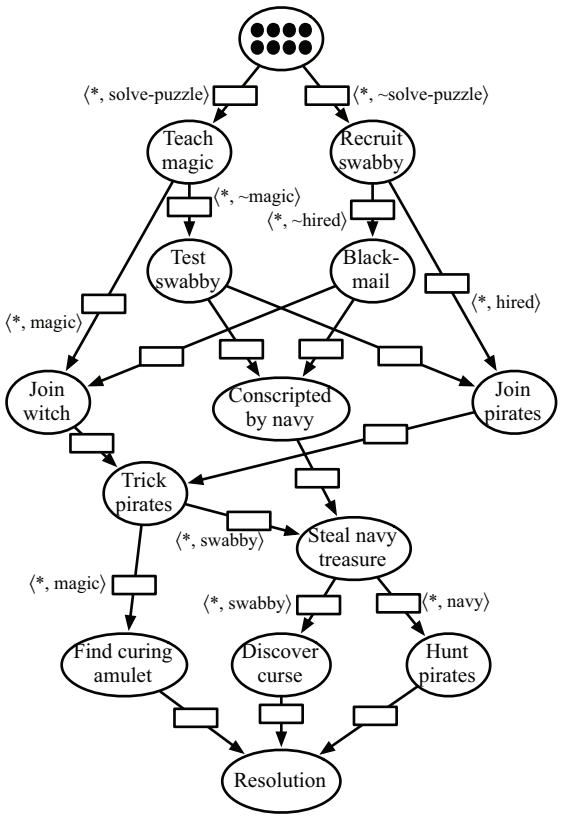

Figure 1: An example Petri Net story. Circles are scenes, tokens are players, and boxes are transitions.

Story Execution

When a scene meets a minimum threshold of tokens, it is marked ready for execution. Unlike typical executions of CPNs, execution does not occur immediately. Instead the system verifies whether the preconditions of "ready" scenes hold for the current world state. This step is critical because the Petri Net models the author's ideal progression but the world may actually have changed in ways unanticipated by the author; e.g., not captured by an explicitly authored branch. There are two reasons this can happen. First, players change the world in unanticipated ways. Second, scenes may "fail"—they terminate without achieving all their effects.

The scene verification step proves that a scene *can* execute because its preconditions are true in the current world state or can be made true given the current world state. If the proof fails, the execution engine must take drastic measures to reconcile the Petri Net with the state of the world so that player experiences can continue. The story execution algorithm is given in Figure 2; the core parts of the algorithm, scene verification and story repair are described below.

Scene Verification Once a scene is ready, MUSE verifies that scenes meet their applicability conditions by building a partial-order plan comprised of new scenes not in the Petri Net. In this sense, a sound and complete plan is a proof that any given scene can be executed barring any further unanticipated changes to the world. Scenes ready for execution are placed into the plan, which may be empty or a complete plan from a previous iteration, and the planner in invoked to attempt to find a sound and complete partial-order plan. If an exception has occurred, then the current world state will not support the preconditions of one or more ready scenes, requiring the planner to instantiate new scenes that bridge between the current world state and the new scenes.

```
let plan ← the empty plan
let state ← current world state
while true do
    Wait for a scene to terminate
    Remove terminated scenes from plan
    state ← UpdateWorldState()
    if there are exceptions then
        plan ← Repair(plan, state)
    UpdatePetriNet()
    if there are scenes ready then
        foreach scene s_i ∈ Ready do
            Add s_i to plan
        let p_new ← Plan(plan, state)
        if p_new ≠ ∅ then
            foreach s_j in p_new with no scenes ordered
            earlier do
                StartExecuting(s_j)
            plan ← p_new
        else
            plan ← Repair(plan, state)
```

Figure 2: The scene execution algorithm.

We use *refinement search* (Weld 1994) to construct the plan. A refinement search planner takes a plan (empty or otherwise) and searches for a sequence of modifications that restores the soundness of the plan. In particular, a plan is not sound if it has an *open precondition flaw*—a precondition of a scene is not supported by the current world state or by the effect of another, temporally prior scene. A data structure called a *causal link*, denoted $s_i \xrightarrow{c} s_j$, records when precondition c of scene s_j is satisfied by an effect of scene s_i (or the initial state).

The scenes from the Petri Net will wait until all of its preconditions are met by the executing plan. Scenes inserted by the planner may involve players directly or may be *maintenance scenes* in which NPCs are directed to change the world "behind the scenes," for example having an NPC replace a missing prop. When newly planned scenes include players, token locations are not adjusted, they remain in place with the understanding that any other planned scenes are meant to restore the applicability of the original scene.

The process of adding new scenes lends robustness to the authored story. By instantiating scenes to bridge between the unexpectedly modified world state and scenes from the CPN, the system is effectively authoring new branches to the story module. Thus the MUSE execution algorithm augments the author's intentions by returning the world state to one intended by the author so that the story can continue.

Story Repair Scene verification is a process whereby changes to the world state impact scenes as they become ready. When a plan is in existence, further exceptions can

also derail the plan; players can cause changes to the world state that will negate causal links in the plan.

Instead of rebuilding the plan from scratch, we use the following plan repair approach adapted from Riedl et al. (2008) to remove elements of the plan that are no longer necessary due to world changes and to graft new scenes into the plan that restore plan soundness. This process is a form of continuous planning, which is, on average, more efficient than replanning from scratch and less likely to produce *dead ends*—executed scenes that never build toward the story conclusion.

First, we distinguish between scenes added to the plan by the Petri Net, S_{Petri}, and scenes instantiated by the planner, S_{Plan}. There are some number of exceptions, represented as causal links no longer supported by the current world state and the scenes they point to. The system applies each of the following repair strategies in sequence until one returns a sound plan.

1. Remove the threatened causal links. Invoke planner to satisfy the now unsatisfied preconditions.
2. Remove the threatened links, the threatened scenes, and any scenes in S_{Plan} that are only on causal chains leading through threatened scenes. Invoke the planner to satisfy any unsatisfied preconditions that result.
3. Remove the threatened links, the threatened scenes, and any scenes in S_{Plan} or S_{Petri} that are only on causal chains leading through threatened scenes. Invoke the planner to satisfy any unsatisfied preconditions that result. Advance tokens in any scene in S_{Petri} that were removed.

The first two strategies attempt to return the world state to one that supports the author's intended story flow.

The third strategy handles the case, as seen in the preliminary study, where authored scenes are bypassed. A scene in S_{Petri} is removed if its effects are preventing other scenes in S_{Petri} from becoming provably executable due to unanticipated changes in the world. If a scene in the Petri Net is bypassed, what happens to the tokens located in that scene? The tokens must be relocated at another scene. The execution engine simply forces the tokens to transition to subsequent scenes. Tokens can always move due to the constraints we impose on transition structure. Forcing tokens to transition will result in one or more subsequent scenes in the Petri Net becoming "ready." It is likely that those scenes will have preconditions that cannot be satisfied by the current world state which will be handled by scene verification and subsequent calls to the story repair mechanism. Strategy 3 is analogous to creating a new branch in the CPN, duplicating human directors' last-ditch efforts to save the story.

Scene Execution A scene is executed when it has a minimum number of tokens *and* all of its preconditions are satisfied by the current world state. That is, there is a valid execution plan and there are no scenes temporally ordered earlier. It may be the case that more than one scene executes in parallel, which is allowed as long as there are no shared resources (NPCs, items, players, etc.) between parallel scenes (Knoblock 1994). If there are shared resources, then one is randomly selected to go first and the others are put on hold until the first completes.

From the perspective of the execution system, scene execution means a message is sent to the NPCs participating in the scene. In the case of our preliminary study, NPCs were human confederate actors, however they could be autonomous agents. Implementation of autonomous character agents is beyond the scope of the paper, but agents built on reactive teleological frameworks such as those used in FAÇADE (Mateas and Stern 2003) and IN-TALE (Riedl et al. 2008) are most appropriate.

Scenes terminate when an adjudication routine—usually built into the NPCs—believes that all the effects of the scene have been achieved. Scenes may also be terminated when the adjudication routine believes that one or more of the effects will not be achieved. This is tantamount to failing the scene.

Authoring

Petri nets, with the constraints we impose, provides a degree of flexibility for authors to express their creativity without requiring authors to conceptualize their story designs as partial-order plans. A Petri Net does impose some structure onto the creative process in terms of breaking the story into scenes and transitions. We found this to be similar to the way our expert director preferred to break up the interactive multiplayer experience to account for different narrative arcs for players. The Petri Net, as we have formalized it, allows the author to think about the story experience from the perspective of the players and the sequence of scenes players will encounter. The Petri Net formalisms also enables branching and scene alternatives, as shown in Figure 1. The execution system will handle the cases where exceptions are not caught by pre-authored transitions.

For robust execution, MUSE authors must also provide a *domain theory*, a description of how the world *can* change. In this case, the domain theory is a set of extra scene templates that, like scenes in the Petri Net, provide preconditions and effects in the form of predicates. These extra scenes are instantiated during scene verification and story repair as means of restoring the Petri Net.

We trained the director from the preliminary study to author interactive narratives using the MUSE representation, consisting of a Petri Net of scenes and a domain library of extra scenes. The human director authored highly specific scenes in the Petri Net (e.g., `distract-guard-and-crack-safe (thief:Ethan, players?, safe1, guard:Chris)`) and general scenes for the domain library (e.g., `steal(`*thief*`?, `*players*`?, `*object*`?, `*owner*`?)`), where question marks denote variables to be bound when the plan is created. The general nature of the domain library created by the director suggests reusability in the sense that many distinct MUSE stories could be authored alongside the same domain library. The only requirements for this is consistent use of predicates.

Finally, an initial state must be provided that describes the characters, places, and objects specific to the MUSE story, as well as relationships between them all. These are used when instantiating general scenes. An ontology of generic proposition types enables authors to construct and describe

unique entities in predicate terms that match the preconditions and effects of scene templates in the domain theory. The initial state is another way in which reusability of the domain is achieved; when scenes are the same, many stories can be told about different characters, places, and objects.

Conclusions

The MUSE approach was developed to directly address the core requirements of multiplayer differentiability, authorability, and robustness for multiplayer interactive narratives in online or real worlds. In particular, we modeled the way the expert director/storyteller focused on individual player trajectories through a space of scenes in alternate reality games.

We note that stories can, and will, derail in multiplayer real and online environments affording rich player expression. Robustness of the interactive narrative experience can thus be achieved through a partnership between human authorial ability and an intelligent system capable of restoring authored scenes to executability. Given a robust drama manager using our story representation, an drama manager can provide players the ability to express their agency—including the ability to perform actions that may derail the story—while managing individual narrative experiences and the overall group social experience. Multiplayer experiences will continue to grow in importance. For these systems to scale, they must allow authors the ability to craft engaging experiences for all players, individually and collectively, without reducing players' abilities to be expressive.

References

Balas, D.; Brom, C.; Abonyi, A.; and Gemrot, J. 2008. Hierarchical petri nets for story plots featuring virtual humans. In *Proc. of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Barber, H., and Kudenko, D. 2007. Dynamic generation of dilemma-based interactive narratives. In *Proceedings of the 3rd Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Barve, C.; Hajarnis, S.; Karnik, S.; and Riedl, M. O. 2010. Wequest: A mobile alternate reality gaming platform and intelligent end-user authoring tool. In *Proceedings of the 6th Conference on Artificial Intelligence for Interactive Digital Entertainment Conference*.

Bruckman, A. 1990. The combinatorics of storytelling: Mystery train interactive. Unpublished manuscript.

Jensen, K. 1996. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 1. Springer, 2nd edition.

Kelso, M.; Weyhrauch, P.; and Bates, J. 1993. Dramatic presence. *Presence: The Journal of Teleoperators and Virtual Environments* 2(1):1–15.

Knoblock, C. 1994. Generating parallel execution plans with a partial-order planner. In *Proceedings of the 2nd International Conference on Artificial Intelligence and Planning Systems*, 98–103.

Laurel, B. 1986. *Toward the Design of a Computer-Based Interactive Fantasy System*. Ph.D. Dissertation, Ohio State University.

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*.

Nelson, M.; Mateas, M.; Roberts, D. L.; and Isbell, C. L. 2006. Declarative optimization-based drama management in interactive fiction. *IEEE Computer Graphics and Applications* 26(3):30–39.

Niehaus, J., and Riedl, M. O. 2009. Scenario adaptation: An approach to customizing computer-based training games and simulations. In *Proc. of the 2009 AIED Workshop on Intelligent Educational Games*.

Porteous, J., and Cavazza, M. 2009. Controlling narrative generation with planning trajectories: the role of constraints. In *Proceedings of the 2nd International Conference on Interactive Digital Storytelling*.

Raybourn, E. M.; Deagle, E.; Mendini, K.; and Heneghan, J. 2005. Adaptive thinking and leadership simulation game training for special forces officers. In *Proceedings of the 2005 Interservice/Industry Training, Simulation and Education Conference*.

Riedl, M. O.; Stern, A.; Dini, D. M.; and Alderman, J. M. 2008. Dynamic experience management in virtual worlds for entertainment, education, and training. *International Transactions on System Science and Applications* 3(1):23–42.

Riedl, M. O.; Saretto, C.; and Young, R. M. 2003. Managing interaction between users and agents in a multi-agent storytelling environment. In *Proceedings of the 2nd International Conference on Autonomous Agents and Multi-Agent Systems*.

Sharma, M.; Ontañón, S.; Mehta, M.; and Ram, A. 2010. Drama management and player modeling for interactive fiction games. *Computational Intelligence* 26(2):183–211.

Trabasso, T., and van den Broek, P. 1985. Causal thinking and the representation of narrative events. *Journal of Memory and Language* 24:612–630.

Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15:27–61.

Weyhrauch, P. 1997. *Guiding Interactive Fiction*. Ph.D. Dissertation, Carnegie Mellon University.

Winegarden, J., and Young, R. 2006. Distributed interactive narrative planning. In *Proceedings of the 2006 AAAI Spring Symposium on Distributed Plan and Schedule Management*.

Young, R. M.; Riedl, M. O.; Branly, M.; Jhala, A.; Martin, R.; and Saretto, C. 2004. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development* 1:51–70.