Reinforcement Learning for Declarative Optimization-Based Drama Management

Mark J. Nelson, David L. Roberts, Charles L. Isbell, Jr., Michael Mateas
College of Computing
Georgia Institute of Technology
Atlanta, Georgia, USA
{mnelson, robertsd, isbell, michaelm}@cc.gatech.edu

ABSTRACT

A long-standing challenge in interactive entertainment is the creation of story-based games with dynamically responsive story-lines. Such games are populated by multiple objects and autonomous characters, and must provide a coherent story experience while giving the player freedom of action. To maintain coherence, the game author must provide for modifying the world in reaction to the player's actions, directing agents to act in particular ways (overriding or modulating their autonomy), or causing inanimate objects to reconfigure themselves "behind the player's back".

Declarative optimization-based drama management is one mechanism for allowing the game author to specify a drama manager (DM) to coordinate these modifications, along with a story the DM should aim for. The premise is that the author can easily describe the salient properties of the story while leaving it to the DM to react to the player and direct agent actions. Although promising, early search-based approaches have been shown to scale poorly. Here, we improve upon the state of the art by using reinforcement learning and a novel training paradigm to build an adaptive DM that manages the tradeoff between exploration and story coherence. We present results on two games and compare our performance with other approaches.

1. INTRODUCTION AND MOTIVATION

We are interested in adding narrative structure to large open-world games in a way that is not pre-scripted and allows the player to interact with and influence the story. Many modern games have rich, non-linear plotlines, with multiple endings, complex story branching and merging, and multiple subplots. We would like the player to be able to have a significant impact on what happens, rather than following along with a prewritten script, or being in control at only a few decision points. However, in contrast to completely open games, the story that results should exhibit global narrative coherence and reflect the game author's aesthetic goals.

Traditionally, story in games is guided by local triggers. Progress in a linear story depends solely on how much of the story has unfolded. In slightly more complex situations, the author can specify condition-action rules (*e.g.* "`if` the player is in the room `and if` the player is carrying a gun, `then` have the non-player character (NPC) hide behind the counter"). The author must specify rules for every possible combination of conditions, a tedious (and intractable) burden for stories of any complexity. Further, the author will find it difficult to achieve a globally cohesive, pleasing story purely from local rules.

This difficulty has been recognized for some time, and grows worse as players demand both true interactive agency and rich stories. One solution is a *drama manager* that watches a story progress, reconfiguring the world to fulfill the author's goals. A drama manager may notice a player moving along a currently undesirable subplot, and attempt to lure the player toward a better one. To do so, it requests that one of the agents in the world take some action; for example, it may instruct an NPC to start a conversation with the player. It may even make requests of objects that appear inert to the player, such as telling a door to lock itself (of course, reconfiguring inert objects must be done carefully without the player noticing). Thus the drama manager acts as a central director of the agents in the game world, making requests of them when necessary to maintain a globally coherent story-line.

We base our work on a specific framework in which a story is represented as a sequence of discrete *plot points*, and the drama manager has a set of requests it can make of agents, which we call *DM actions*.

Plot points are important possible events in the story, such as a player discovering information or an object. DM actions are any requests for which the game author has implemented an agent that can respond to the request. Plot points have a set of (boolean) prerequisites defined as other plot points. For example, the plot point *found book* may have the prerequisite *found key*, if it's hidden in a locked room. Prerequisites restrict the set of plot-point sequences—and thus stories—to those that are physically possible. Note that in nonlinear games with many plot points there are still a large number of possible stories.

Plot points are abstractions of concrete events happening in the game world: For example, in a story that we use as a case study, *Anchorhead*, the player can find a safe in an old mansion. Finding the safe involves a number of

Preprint. Submitted to the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06), May 8–12, 2006, Hakodate, Japan.

concrete actions, such as entering the room in which it's located, reading a description of the room, possibly examining various objects in the room, searching the bookshelf on the wall, and finally picking up the book behind which the safe is hidden. The drama manager does not need to know the details of all these actions, since most do not impact the progression of the story; instead, it is only notified of the abstract plot point *found safe*.

Drama-manager actions are similarly abstract, but correspond to sequences of concrete actions in the game world. For example, the DM action *hint safe location* may result in a complex series of concrete events, in which an NPC tries to draw the player into a conversation and let slip a hint. To the drama manager, what matters is the expected outcome of the request, so DM actions are given as an ordered pair (*action*, *plot point*), where *action* is one of: *cause*, *deny*, *temporarily deny*, *reenable*, or *hint*. These actions cause the plot point in question to, respectively, either: happen immediately (*e.g.* an NPC gives the player an item); become impossible for the duration (*e.g.* inconspicuously make an item disappear from the world entirely before the player has noticed it); become temporarily impossible (*e.g.* tell an NPC to refuse to speak to the player for now); become possible again (*e.g.* tell the NPC to be talkative again); or become more likely by leading the player towards the plot point (*e.g.* tell an NPC to hint about an item's location). Actions have constraints. They must be consistent (a plot point cannot be *reenabled* unless it has previously been *temporarily denied*), and there can also be constraints on when, from the author's perspective, the action makes sense (*e.g.* an action that *causes* something should only be taken at points in the story when it is plausible to cause its target plot point).

Finally, the author provides an evaluation function that takes a sequence of plot points and the history of drama-manager actions, and rates the completed story. This is the central feature of what we call *declarative optimization-based drama management* (DODM): The author specifies *what* constitutes a good story, and leaves to the drama manager the problem of *how* to bring it about in the complex environment of a continually-progressing game.

To specify an evaluation function, the author will generally annotate plot points or DM actions with some information and then specify a *feature* using that information, which is a numerical score rating how much a particular story exhibits some desired property. The final evaluation function is a weighted sum of these features, in accordance with the author's judgment of their importance. For example, the author may annotate plot points with a the subplot they belong to, and assign a higher score to stories where the player progresses along one subplot for several plot points in a row; or, the author may annotate plot points with a description of the knowledge they provide to the player, and assign a higher score to stories where knowledge unfolds gradually and climaxes in a burst of revelations near the end (*e.g.* in a mystery story). Actions can be annotated with a judgment of how likely the player is to sense that the drama manager is fiddling with the world; stories would then be assigned scores in inverse proportion to how much total manipulation took place. These desired features may in many cases conflict in surprising and constantly changing ways; the drama manager is responsible for making these complex tradeoffs.

In short, the problem of drama management is to optimize the use of the available drama-manager actions in response to a player's actions, given the set of plot points and an evaluation function. This is well-modeled as a reinforcement learning (RL) problem. RL algorithms attempt to learn a policy that, given a world modeled by states and a (possibly unknown) transition function between states, specifies an action to take in each state in a manner that maximizes expected evaluation according to some function. In this case, state is specified by the sequence of plot points and drama-manager actions that have happened so far (order is important); actions are the drama-manager actions plus the null action (do nothing); transitions are the likelihood of plot points occurring in any given state, and are given by the combination of prerequisites for a plot point, the player, and any influence from drama-manager actions; and the evaluation function is the one provided by the author. Although nicely defined, this is a complicated reinforcement-learning problem: the player is difficult to model and highly stochastic, and the state space is extremely large.

In the rest of this paper, we present our approach to this problem. We review earlier search-based techniques, discuss our application of temporal-difference (TD) learning methods to the problem, and show how our novel approach to training, *self-adversarial / self-cooperative exploration* (SASCE), outperforms search-based methods and improves the performance of TD methods.

2. RELATED WORK

Using a drama manager to guide interactive drama was first proposed by Laurel [3]. The story and drama-manager formalism we use—and treating drama management as an optimization problem—was proposed by Bates as *search-based drama management* (SBDM) [1]. SBDM formulates the drama management problem similarly to a minimax game-tree search problem as has been used for games like chess; however, because the player in interactive drama is not truly an opponent, expectimax search is used, alternating maximizing nodes where the drama manager chooses its best available action with expectation-calculating nodes where the player is assumed to act according to some probabilistic model.

Weyhrauch [12] applied SBDM to a simplified version of the Infocom interactive fiction *Deadline*, named *Tea for Three*. Full-depth search is intractable, so he used shallow searches with static cutoff values calculated by sampling a small number of possible stories that could follow the cutoff and averaging their evaluations, achieving impressive results on *Tea for Three*. Lamstein & Mateas [2] proposed reviving the technique, but work by Nelson & Mateas [7] on a different story, *Anchorhead*, showed that the impressive performance of Weyhrauch's limited-depth sampling search did not generalize to other stories.

In formulating drama management as an RL problem, temporal-difference (TD) learning [9] seems a natural candidate algorithm. In particular, the approach used by TDGammon [10] to learn a policy for playing backgammon as well as the best human players [11] seems a useful starting point. Some have argued (to some controversy) that these results are due in part to the particular stochastic properties of backgammon [8]. As we shall see, we address this concern by modifying the stochastic properties of the training regimen.

There have also been several other drama management

systems that take quite different approaches.[1]

The Mimesis architecture [13] constructs story plans for real-time virtual worlds. The generated plans are annotated with a rich causal structure, and the system monitors for player actions that might threated causal links, replanning or preventing player actions if a threat is detected. This approach differs substantially from ours in that authors specify concrete goals for the planner to achieve as part of the story, while an author in DODM specifies abstract and possibly conflicting features that the story as a whole should possess.

The Interactive Drama Architecture [4] takes a prewritten plot and tries to keep the player on the plot by taking corrective action according to a state-machine model of likely player behavior. This addresses a much different problem than ours, since we're interested in letting players' actions substantially affect the overall plot, rather than just incorporating their actions into variants of a prewritten plot.

The beat-based drama manager in *Façade* [6] sequences small pieces of story progression called dramatic beats, constructing the story out of these directly-chosen pieces. This is substantially different from the DODM approach, which for the most part lets the story run itself in the traditional way, stepping in to instruct the agents with DM actions only when it seems necessary. We hypothesize that this makes beat-based drama management better suited to tight story structures, where ideally all activity in the story world contributes to the story, while DODM lends itself more naturally to more open-ended games, especially in large worlds that may include much player activity not directly relevant to the story.

3. METHODS

In this paper, we focus on quantitatively evaluating the performance of the optimization component of DODM. Given a story world modeled by plot points, a set of available drama-manager actions, and an author-supplied evaluation function, we can determine to what extent the drama manager succeeds in maximizing story quality according to that evaluation function. To do so, we use the drama manager to guide simulated games many times, score each resulting game using the evaluation function, and plot the distribution of scores on a histogram. For a baseline distribution, we run the same simulations without a drama manager (equivalent to a null drama manager that never takes any DM actions). If the drama manager is successful, its distribution will be generally skewed towards the right compared to the baseline: high-quality stories will appear more often, and poor-quality stories will happen infrequently if at all. For the simulations, we use a fairly naive player model (proposed by Weyhrauch) that assumes the player is exploring the world; the player chooses uniformly from available plot points (plot points whose prerequisites are satisfied), with increased probability of doing something the drama manager has hinted at.

We compare three approaches to optimizing the selection of DM actions: the limited-depth sampling search (SAS+) proposed by Weyhrauch; standard temporal-difference (TD) learning; and TD learning with our modified training regimen (SASCE). We evaluate our results on two different stories: one a variant of the *Tea for Three* story used by Weyhrauch, and the other the *Anchorhead* story used by Nelson & Mateas; we also perform more detailed analysis on a subset of *Anchorhead*.

3.1 SAS+

Weyhrauch's search-based drama manager [12] chooses actions by performing expectimax search, a modification of standard minimax game-tree search in which the "opponent" (in this case the player) is assumed to be acting probabilistically according to the player model rather than adversarially. Since full-depth search in even fairly small stories is intractable, he proposes a sampling search, SAS+. Full-width search is performed to a fixed cutoff depth; from each node at the cutoff, a fixed number of possible plot-point sequences that could follow that node are sampled and scored; and the average of their scores is the evaluation of that node. To avoid dead ends, temporarily-denied plot points are reenabled prior to sampling, but the drama manager otherwise takes no action during sampling.

3.2 TD learning

Temporal-difference learning [9] is a reinforcement-learning technique that, over many thousands of simulated games, estimates the values of story states (a story state is either a partly- or fully-completed story). For completed stories, the state value is simply what the evaluation function says it is; for partial stories, it is the expected value of the complete story (*i.e.* the average of all stories that are continuations of that partial story, weighted according to likelihood). The drama manager's policy is then to simply pick whichever DM action leads to the highest-valued successor state.

A full description of TD learning is beyond the scope of this paper, but the general idea is that as a simulation run progresses, the value of each state encountered is estimated based on the current estimate of its successor. Since at the end of each story we can use the evaluation function to get the real value for the last state, the real values propagate backwards, and all states' estimates should eventually converge to the true values. Since there are far too many possible story states to store the value estimates in a lookup table, we follow Tesauro [10] in training a neural network as a function approximator to estimate a function from states to their values.

In story-based games, the order of plot points is of course important, so the state cannot simply include what has happened, but also the order in which it happened. We encoded state as an $n \times n$ precedence matrix, where n is the number of plot points and $m_{i,j}$ indicates whether plot point i happened after plot point j.[2] For example, in a story where there are four possible plot points, and in which plot points 3, 4, and 2 have occurred so far (in that order), the precedence matrix would look like:

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

[1]The interested reader is directed to [5] for a more complete review of the drama-management literature (as of 1997).

[2]This bears some resemblance to the more common bigram encoding, which specifies for each pair of events i, j whether that *sequence* appeared in the history. The precedence matrix trades off some information about exact sequences for a gain in information about global ordering.

3.3 SASCE

Self-adversarial / self-cooperative exploration (SASCE) is our modified training regimen for TD learning. Our experiments on TD learning both with and without SASCE were identical except for the player model used for training. Typically in machine learning we assume that training data is drawn from the same distribution as the test data. In this domain the TD learner would therefore be trained on a player similar to the one it is expected to encounter when deployed. In our case, this is the player exploring the world.

Instead, the SASCE player model uses the drama manager's current value function to select its actions. Because the SASCE player model uses the agent's state-value estimates, it can choose to act cooperatively or adversarially. A cooperative player model would select actions that put the player in the highest-rated subsequent state, while the adversarial player would move to lower-rated states. The basic idea behind SASCE is to convert the asymmetric non-adversarial drama management problem into a *pseudo-adversarial* or *pseudo-cooperative* problem. This feedback loop is intended to force meaningful exploration of the state space.

We implemented the player model to select states based on an exponential distribution:

$$p(s_i) = \frac{e^{\alpha\beta(s_i)V(s_i)}}{\sum_{s_j \in S} e^{\alpha\beta(s_j)V(s_j)}}.$$

Here s is a state; $V(s)$ is the agent's current state-value estimate; α is a parameter controlling the "desire" of the model to cooperate (large positive values are more strongly cooperative and large negative more adversarial); and $\beta(s)$ is function that responds to the "desires" of the drama manager. In our case, $\beta(s) = 1.0$ for all states except for those the manager has explicitly attempted to increase the likelihood of (*e.g.* by providing a hint). In those cases, $\beta(s)$ is some constant $c_a > 1$ associated with action a.

In practice, we train using a *mixed strategy*. The mixed-strategy player has three parameters. It has both co-operative and adversarial values for α and a probability (q) that the player will use the adversarial value of α. The goal of the mixed strategy is to keep from learning a worst-case policy: Since we don't expect the player to *actually* be acting adversarially, we don't want the policy to optimize only for that case.[3]

4. RESULTS

We first present results on *Tea for Three* and a subset[4] of *Anchorhead*, demonstrating that TD learning outperforms SAS+ on complex domains, and further that using SASCE improves TD learning's performance on both relatively simple and complex domains. We analyze this subset of *Anchorhead* extensively to understand why the policies behave as they do. We then present some results on a larger portion[5] of *Anchorhead* in which none of the policies perform well, but demonstrate that adding more DM actions improves performance markedly, illustrating some potential authorship pitfalls.

[3]The reader may wish to improve upon our results by using our mixed strategy only as a bootstrapping mechanism. Although space limitations preclude a lengthy discussion, an attempt to simulate this by gradually annealing α and q towards an exploring player provided no extra benefit.

[4]The subset includes all the plot points from one major subplot, and all DM actions relevant to those plot points.

[5]The larger portion is the one used by Nelson & Mateas [7], itself a subset of the very large original interactive fiction piece.

	Tea for Three		*Anchorhead* subset	
Method	Mean	Median	Mean	Median
SAS+	**96.3%**	**96.7%**	42.5%	50.5%
TD	81.2%	81.8%	80.4%	70.5%
TD/SASCE	91.3%	93.3%	**82.8%**	**83.6%**

Table 1: Summary of results using SAS+ search and TD learning with and without SASCE. Numbers are in percentiles relative to the non-drama-managed distribution of story qualities (see text).

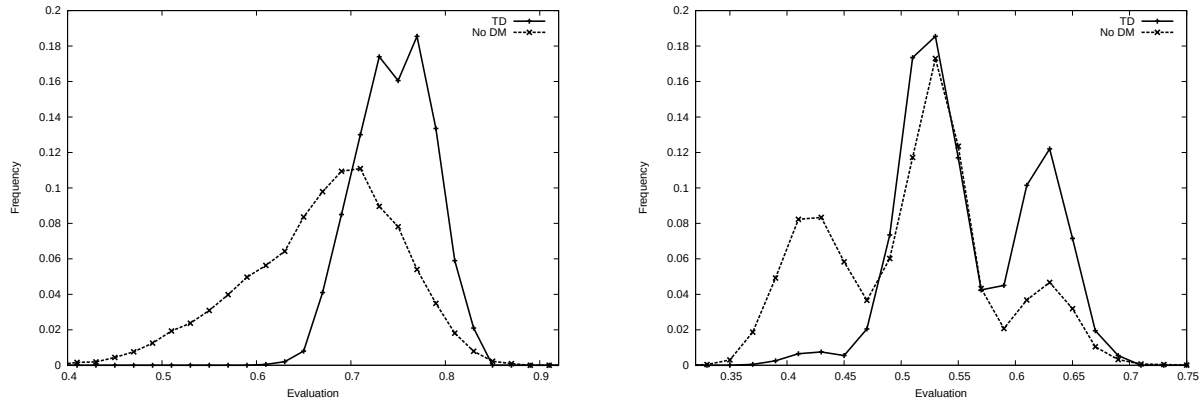
4.1 *Tea for Three* and an *Anchorhead* subset

Table 1 summarizes the performance of SAS+ sampling search, TD learning trained conventionally, and TD learning trained using the SASCE training regimen, each evaluated on both *Tea for Three* and a subset of *Anchorhead*. Performance is summarized by calculating the mean and median story quality of each drama-managed distribution, and reporting that in terms of a percentile of the non-drama-managed distribution (*e.g.* a percentile of 70% for the mean would say that the average story quality using drama management is better than 70% of stories without using drama management).

There are three interesting things to note. First and foremost, training TD using SASCE consistently improved its performance by all measures on both stories. Secondly, while search does marginally better than SASCE on the simpler *Tea for Three* story, its performance degrades significantly on the *Anchorhead* subset—which has a more complicated relationship between story structure and evaluation—while the RL methods continue to perform well. Finally, SASCE greatly improves median performance on the *Anchorhead* subset in comparison to conventionally-trained TD learning.

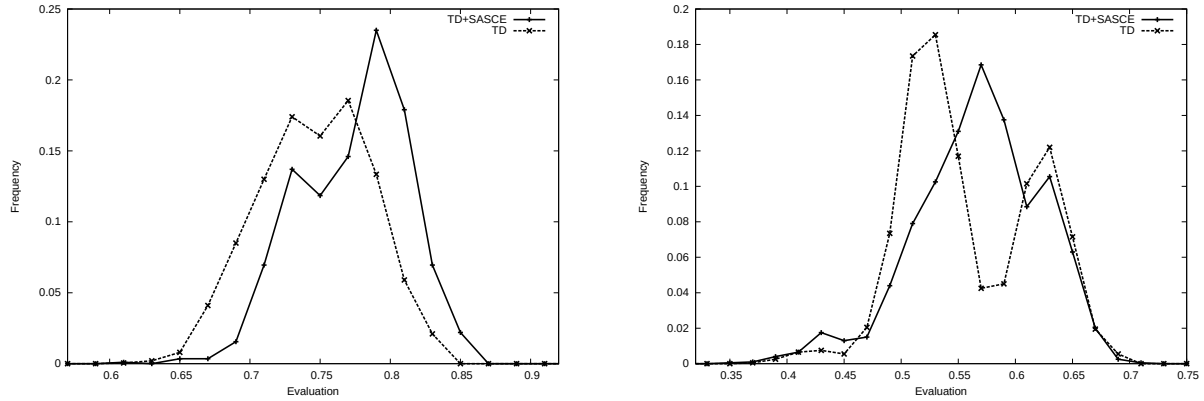
Figure 1 shows the detailed distributions on each story both without drama management and using TD learning without SASCE. Notice in the non-drama-managed distributions that the stories are qualitatively quite different: *Tea for Three* has a relatively smooth unimodal distribution, while the *Anchorhead* subset is distinctly trimodal. This suggests that there are points in the *Anchorhead* subset that have significant influence on the ultimate outcome. It is at these points that SASCE learns to make better decisions. This also suggests why search performs so poorly in the *Anchorhead*-subset domain: its relatively short horizon doesn't allow it to effectively make long-term global decisions. As a result it only performs well in "easy" domains where local decisions can result in globally good results, as noted in [7]. We were able to reproduce these results, but also show that reinforcement learning is able to make global decisions in both types of domains effectively.

While not shown in a figure, the story-quality distribution using search on *Anchorhead* remains trimodal like the distribution of randomly-generated stories, lending support to the hypothesis that search fails to find the important deci-



(a) *Tea for Three* (b) *Anchorhead* subset

Figure 1: Story-quality distributions for conventionally-trained TD learning versus no drama management.



(a) *Tea for Three* (b) *Anchorhead* subset

Figure 2: Story-quality distributions for TD learning trained using SASCE versus trained conventionally.

sion points that steer between the different types of stories. By contrast, the TD-learned policy's distribution is bimodal and almost entirely missing the stories that form the lowest of the three modes (see Figure 1(b)), indicating that it successfully learns how to avoid the worst batch of stories.

The policy trained with SASCE does even better, with a unimodal distribution that is shifted higher (see Figure 2(b)), indicating that it is not missing any major decision points. To give a more detailed idea of how SASCE improves TD learning, Figure 2 shows the full distributions of story qualities using each method on both stories we evaluated. Remember, qualitatively the goal is to shift the distribution upwards as far as possible, reducing the frequency of lower-scored stories and increasing that of higher-scored stories. Notice how in Figure 2(a), SASCE is consistently better: On the lower side of the distribution (below the mean) its frequencies are lower, while on the higher side they're higher. In Figure 2(b), the improvement takes on a different shape, keeping roughly the same proportion of highly-scored stories, but increasing the quality of the lower- and middle-scored stories significantly, pushing the main peak (and the median) towards the upper end of the range.

Recall that, using SASCE, a player model can be adversarial (negative α), cooperative (positive α), or mixed (choosing an adversarial player with probability q, and a cooperative player with probability $1 - q$). Interestingly, all mixed models except the fully cooperative player ($q = 0.0$) improved TD learning, both quantitatively and qualitatively. In the examples shown here, as in most of our experiments, training on the mixed player gave the best performance, as shown in Figure 3, although the mean and median are not significantly different. Training on the fully cooperative player consistently gave the worst performance. All types of SASCE players display a roughly unimodal histogram, in stark contrast to the bimodal histogram seen in conventionally-trained TD learning, and the trimodal distri-

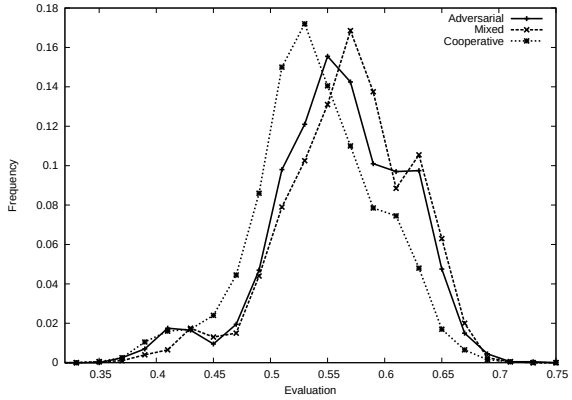


Figure 3: Story-quality distributions on the *Anchorhead* subset using different values of q for training. The three values shown are $q = 0.0$ (cooperative), $q = 0.5$ (mixed), and $q = 1.0$ (adversarial).

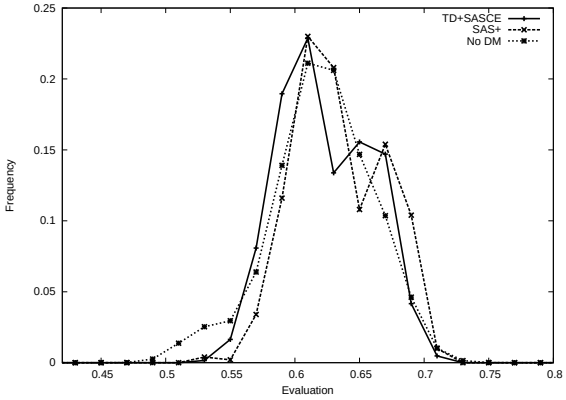


Figure 4: Story-quality distributions on *Anchorhead* for SASCE-trained TD learning, SAS+ search, and no drama management.

Adversarial			Mixed		TD
Mixed	TD	Search	TD	Search	Search
5.6	5.1	1.4	4.8	1.4	1.4

Table 2: Average number of plot points occurring in the *Anchorhead*-subset story before policies disagree on which action to take.

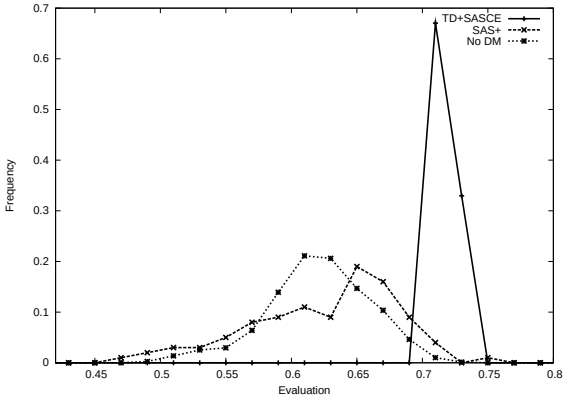


Figure 5: Story-quality distributions on *Anchorhead* with synthetic actions (see text) for SASCE-trained TD learning, SAS+ search, and no drama management.

bution of the unmanaged stories (Figure 1(b)). This lends some further support to our hypothesis that the modes are due to distinct decision points.

Intuitively, training against an (even occasionally) adversarial player forces the drama manager to learn how to maximize the worst-case performance. That is, the manager learns how to take actions that will prevent the player from arriving in a state where it can then easily do harm to the story. This proves useful for a wide variety of actual player types, including players who are exploring. A cooperative player acts as an antidote for those cases where such states can also lead to remarkably good outcomes.

It is worthwhile to try to understand in more detail how the policies learned by the various methods differ. To that end, we use them simultaneously to guide our simulated player until they reach a point at which they disagree on which action to take. Table 2 shows the average number of plot points occurring prior to disagreement for each pair of policies. Search is making different decisions very early. TD without SASCE diverges from SASCE-trained TD around the fifth plot point. Interestingly, the policies trained with adversarial and mixed players diverge not much later, but have much more similar distributions, suggesting that the important decisions have already been made by the fifth or sixth plot point (on average).

Finally, while space prevents a full discussion, it is worth noting that when there was a disagreement between policies on which action to take, we kept track of the plot point that preceded the disagreement and the pair of conflicting actions. We found that there were a small number of plot points after which most of these disagreements occurred, and the disagreements themselves were mostly between a few pairs of conflicting actions. That suggests that there are likely several key decision points directing stories into one of the three major classes in the trimodal story-quality histogram, and these decision points are where the policies differed most noticeably.

4.2 *Anchorhead*

On the larger portion of the *Anchorhead* story, which includes two subplots that may interleave leading to two possible endings, none of the policies do particularly well, as illustrated in Figure 4.

In order to determine whether the policies were performing poorly because of their inability to find a good policy, or because the available DM actions in this state space simply did not permit for a very good policy, we ran a "synthetic" test in which there was a causing, hinting, temp-denying, and reenabling DM action for every plot point ("synthetic" because many of these DM actions could not plausibly be implemented in the real story). This ought to give the drama manager complete control, and indeed as shown in Figure 5,

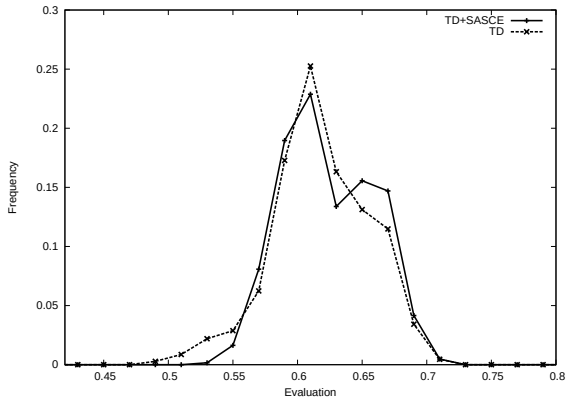


Figure 6: Story-quality distributions on *Anchorhead* for TD learning with and without SASCE.

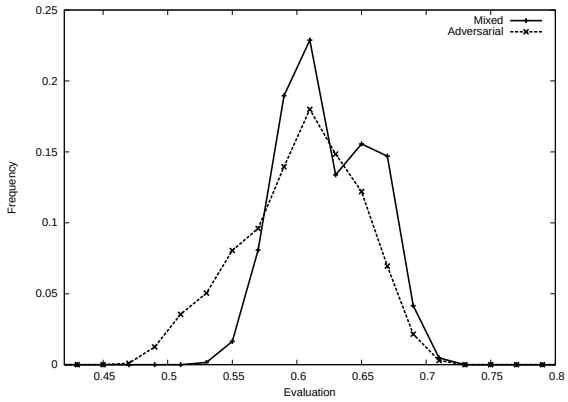


Figure 7: Story-quality distributions on *Anchorhead* for TD learning with mixed-strategy SASCE training ($q = 0.6$ in this case) and completely adversarial SASCE training ($q = 1.0$).

the TD-learned policy is now consistently very good; SAS+ search, however, still does rather poorly, as it is unable to navigate the now even larger state space to make effective use of the added actions. While this does not *prove* that the poor performance without these added actions is due to a lack of sufficient actions, rather than inability to learn a good policy, it does lend some support to that hypothesis, and suggest that the drama manager should be given more actions to work with.

The shape of Figure 5 illustrates a further issue: Although the goal as specified is to maximize story quality, it is not clear that this type of narrow distribution is the sort we actually want, especially since when we examined the plots produced, we found them to be nearly all identical: the drama manager found a very small range of good stories, and used DM actions to cause those stories to always happen. However, we would prefer there be a range of possible stories that could emerge during gameplay, especially if replay value is important. In non-synthetic examples this may turn out to not be an issue most of the time, but it does suggest that perhaps we should extend the evaluation function so that it takes into account the diversity of possible stories as an explicit goal, rather than rating each story in isolation.

Interestingly, SASCE still noticeably improved the performance of TD learning in this case, as illustrated in Figure 6. Furthermore, mixed-strategy training in particular is what improved performance: the completely adversarial version of SASCE did poorly, as shown in Figure 7. These consistent results across all the stories we tested lend some support to our hypotheses that SASCE forces the right type of exploration for this domain, and that a mixed-strategy version of SASCE in particular balances forcing exploration with an adversarial player without learning a policy optimized only for the adversarial case.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a declarative optimization-based drama manager. This style of drama management centrally coordinates the behavior of agents in a game by making requests of them (*DM actions*) in a manner that maximizes story quality according to some author-specified evaluation function. We demonstrated the use of reinforcement learning to learn, ahead of time, policies for taking these actions, rather than performing online search in the time-constrained environment of a real game. We showed that reinforcement learning performs on par with search in simple domains, and far outperforms search in more complex domains. Furthermore, we presented a training method, SASCE, that trains on player models that use the drama manager's current value estimates in such a way as to force exploration useful for this domain; we demonstrated that it improves learning performance in all cases we examined.

One avenue of future work involves a more complete understanding of why training on the sort of player model presented here yields improved performance over training on the same player used for evaluation (a "known correct" player model).

There are other player types in addition to the ones we have explored here. For example, there are players who really want to move through the story quickly, and players who are simply trying to delay ending the game (while an explorer, this type of player is not quite captured by our model). Initial experiments on other players suggest results similar to those presented here, but further investigation is warranted.

Ultimately, we would like to test our learned policies on dozens of actual human players interacting in the concrete world of our chosen stories, and see what distribution this induces in the abstract world of plot point sequences. Unsurprisingly, because players are experiencing concrete events in the story world, what seems like a logical model of player behavior in the abstract world has the potential to be inaccurate. For example, it is reasonable to expect a player to unlock a door shortly after finding a key; however, if the door is in a dark corner of the room, the player may not see it. In the future we plan to build an agent that represents the player in the concrete world rather than in our abstract plot point world so that we can model these types of influences.

In any case, developing and using a wider range of player models for evaluation would allow us to better understand the relationship between the player model used for training and that used for evaluation. In particular, we are interested in how robust policies are when used with different

sorts of players. For example, if we determine that policies trained under varying player models perform better or worse against different player models used for evaluation, we hope to develop techniques to identify in real-time the best policy to use to optimize a given player's experience.

Finally, we are interested in further evaluating the authorial affordances offered by this style of drama management, both in easing the design of large games and in offering possibilities for new types of games difficult or impossible to design without a system of this sort.

Acknowledgments

Work on this project has been supported in part by a National Science Foundation Graduate Research Fellowship and a grant from the Intel Foundation. We also wish to thank Victor Bigio for his help coding and running experiments.

6. REFERENCES

[1] J. Bates. Virtual reality, art, and entertainment. *Presence: The Journal of Teleoperators and Virtual Environments*, 2(1):133–138, 1992.

[2] A. Lamstein and M. Mateas. A search-based drama manager. In *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*, 2004.

[3] B. Laurel. *Toward the Design of a Computer-Based Interactive Fantasy System*. PhD thesis, Drama department, Ohio State University, 1986.

[4] B. Magerko. Story representation and interactive drama. In *Proceedings of the First Annual Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-05)*, 2005.

[5] M. Mateas. An Oz-centric review of interactive drama and believable agents. In M. Woodridge and M. Veloso, editors, *AI Today: Recent Trends and Developments. Lecture Notes in AI 1600*. Springer, Berlin, NY, 1999. First appeared in 1997 as Technical Report CMU-CS-97-156, Computer Science Department, Carnegie Mellon University.

[6] M. Mateas and A. Stern. 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 (TIDSE-03)*, 2003.

[7] M. J. Nelson and M. Mateas. Search-based drama management in the interactive fiction Anchorhead. In *Proceedings of the First Annual Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-05)*, 2005.

[8] J. B. Pollack and A. D. Blair. Why did TD-Gammon work? *Advances in Neural Information Processing Systems*, 9:10–16, 1997.

[9] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[10] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[11] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[12] P. Weyhrauch. *Guiding Interactive Drama*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1997. Technical Report CMU-CS-97-109.

[13] R. M. Young, M. O. Riedl, M. Branly, A. Jhala, R. J. Martin, and C. J. Saretto. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*, 1(1), 2004.