Authoring an Interactive Narrative with Declarative Optimization-Based Drama Management*

Mark J. Nelson[1], **Calvin Ashmore**[2], and **Michael Mateas**[1,2]
[1] College of Computing
[2] School of Literature, Communication, and Culture
Georgia Institute of Technology
Atlanta, Georgia, USA
mnelson@cc.gatech.edu, ashmore@gmail.com, michaelm@cc.gatech.edu

Abstract

Drama managers reconfigure a game in reaction to a player's actions. In declarative optimization-based drama management (DODM), a game's story is abstracted as a sequence of plot points; possible drama manager interventions are abstracted as a set of DM actions. The author defines an function evaluating story quality, and some optimization method (currently reinforcement learning) chooses DM actions so as to maximize expected story quality according to that evaluation function. While previous work has developed this approach at a technical level and discussed its potential applications, no work to date has used DODM to write real games. We report on our experiences designing a game in the Neverwinter Nights engine, entitled *The Guilty*, in which we use DODM to create a dynamic plot that in a previous design iteration we had found difficult to create with other techniques.

Introduction

We're interested in providing tools for game designers to author story structures that move beyond local triggers and quasi-linear plot structure. For a player to have a strong impact on a story, the game must react in appropriate ways to various combinations of behavior. Local triggers require this knowledge to be embedded in the game world, where it is difficult to ensure that the interaction of triggers with the player and each other results in a globally cohesive story in keeping with the author's design goals. We would prefer a decoupling, where the game world encodes game mechanics and another system is responsible for dealing with the plot.

This is a drama manager (DM): A system that watches a story, intervening when necessary to ensure the author's goals are fulfilled. For example, the drama manager might direct a non-player character (NPC) to start a conversation with the player. One type of DM is declarative optimization-based drama management (DODM), in which the story is modeled as a sequence of discrete *plot points*, and the DM has a set of ways it can intervene in the world, termed *DM actions*. The author specifies an *evaluation function* that declaratively encodes his or her judgment of story quality: Given a completed story represented by a sequence of plot points, it assigns a score. The drama management problem is thus an optimization problem: Determine how to use the DM actions so as to maximize expected story quality.

While DODM has been developed on the technical level (Weyhrauch 1997; Nelson *et al.* 2006), its usefulness in authoring real games has not been much evaluated. In this paper, we discuss the design of a new game, *The Guilty*, that uses DODM as an integral part of the design.

Overview of DODM

The author provides DODM three components: a set of plot points, a set of DM actions, and an evaluation function.

A plot point is an abstraction of a concrete event in the game world that's significant enough for the DM to need to know about. When it occurs, the game signals the DM, and waits to see if the DM requests a DM action in response. Generally plot points should either be events whose occurrence might impact the overall quality of the story, or events at which the DM could conceivably want to intervene. These plot points are annotated with prerequisites giving ordering constraints in the game world; for example, *get key* might be a prerequisite for *find book* if the book is in a locked room. They are also annotated with any information the author finds useful in writing an evaluation function; for example, the location they take place in.

A DM action specifies a particular intervention into the game by the drama manager. This is also an abstract entity at the DM level; when the DM signals to the game that it should happen, the game enacts it as some concrete set of events. To the DM, 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: happen immediately (*e.g.* an NPC gives the player an item); become permanently impossible (*e.g.* inconspicuously make an item disappear from the world before the player notices); 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 also have constraints. They must be consistent (a plot point cannot be *reenabled*

*This research is supported by a grant from the Intel Foundation and by the National Science Foundation's Graduate Research Fellowship program.

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 it). They also have annotations; for example, the author may annotate an action with an estimate of how manipulative it might seem.

One issue with both plot points and DM actions is at how high a level to specify them. The simplest solution is to have them at low level: A plot point is triggered directly by the execution of a line of code, and a DM action triggers a piece of code. In many cases it would be more convenient to have higher-level plot points and DM actions. For higher-level plot points, a *recognizer* is needed, to determine when the event in question has happened; similarly, DM actions need an associated *refiner* to turn the abstract action into concrete activity in the game world. More complex recognizers and refiners are more feasible in more "intelligent" underlying game worlds with; for example, if characters in the game are agent-like with some internal state, they can receive DM actions as requests and refine them themselves.

Finally, the author writes an evaluation function encoding his or her goals. The evaluation function takes a completed story, represented by its sequence of plot points and history of drama-manager actions, and assigns it a score. The evaluation function is typically based on a number of features, each of which may be weighted according to the author's preference. Each feature in turn is computed based on the various annotations on plot points and DM actions. This evaluation function is the declarative feature of declarative optimization-based drama management: The author only has to specify what constitutes a good story, not the complex tradeoffs that determine how to bring about such a story.

This produces a well-defined optimization problem: The DM needs to choose the action at each step that maximizes the likelihood of the player experiencing a good story. Currently we use reinforcement learning to learn a policy for choosing DM actions, as detailed by Nelson *et al.* (2006).

The Guilty

The Guilty relies on DODM as an integral part of the design, based partly on our experiences in an earlier design iteration that encountered several fundamental problems.

Plot

The game is set in a small city ruled by a tyrannical lord, in which the player character is the feared and unquestioned captain of the guard. From the beginning it is clear that there are a number of illicit and shady things taking place in the city, about which the player knows nothing, but the player character himself shrugs these things off impassively.

The distinction between the *player*—the human playing the game—and the *player character* he or she is playing is a significant aspect of the design. The player is not really sure what sort of character he or she is playing; in particular, the player does not know at the beginning of the game what took place up until the point that the game started. In addition, the player is not even completely in control during the game: The game is segmented into scenes, and between scenes the character can participate in events that aren't immediately known to the player. One of the goals is to create tension in the usual self-identity that a player feels when playing the role of a character, following the literary tradition of unreliable narrators.

When the player character's partner is found murdered, it is the player's job to investigate and arrest the killer. It is later revealed that the player character was in fact the murderer, who at various points (between scenes) had been working to thwart the player's investigation. The real question is why he did it, and that depends on what happens in the game: The player's actions influence not only the events in the game itself, but what backstory we reveal. Possible reasons for the murder can vary greatly: the player may have had an affair with the victim's widow; the victim could have been an honest cop who was uncovering something shady that the player was involved in; the victim could have been abusive to his widow, with whom the player was secretly in love; and so on.

The major premise of the game is that nearly all parts of the backstory—besides the general setting and the fact that the character did commit murder—are determined incrementally at game time, being fixed only insofar as parts have already been revealed to the player.

Previous design attempt

Our first design used local triggers and ran into significant scaling problems. Since the game requires strong global interaction between events, the triggers need to refer to global variables that are kept updated. These proved nearly impossible to keep track of: Because of the large number of possible combinations, there was no feasible way to manage how clues were placed, restrict what dialogue options were available, and otherwise make concrete the various options for the past history of the character. Even if somehow we had managed to write a fairly small game with this approach, it would've been completely unmaintainable, since even relatively small additions or changes would require us to completely rework a large number of other triggers.

This version also included a karmic system, of the sort currently popular in a number of commercial games (e.g. *Fable*). We maintained several gauges summarizing the player's character—anger, lust, guilt, and so on—and incremented or decremented them when the player performed various actions. Clues and conversation options would then be based on these karma sliders. Restricting the conversation options based on the karma values led to several undesired consequences, since in order to enforce consistency, it caused a positive feedback cycle. The player's behavior in making one decision would add on to, say, the guilt attribute, and the next round of possible options would restrict the non-guilt choices, inevitably locking the player into one slot before having a chance to explore the others.

The karmic system also limited our ability to have different facts in the game be true or false. If the player just worked up to having attributes that implied he was having an affair, but then if the player's attributes changed slightly, the affair would not actually make sense anymore. This re-

quired us to add some consistency checks, but the number of details to keep track of made that approach quickly intractable for the same reasons as with the local triggers.

DODM design and analysis

Designing the game with DODM seems to resolve a number of our previous difficulties. We annotate plot points with a note about what information they convey to the player. For example, some plot points involve the player learning that he did something particular in his past; others vaguely suggest something (but don't definitively commit); and so on. The evaluation function can then score a story based on the sequence of information-revealing episodes, and therefore the DM can use its DM actions to ensure that information is revealed in keeping with our authorial goals.

The most important of our goals is that the story must be coherent. The simplest approach to this would be to fix portions of the backstory whenever information is revealed. To get a more fluid and interesting effect, though, the DM has more flexibility: Information that is revealed can later be shown to have been wrong or somehow misleading (a common occurrence in detective/mystery stories). The DM must still take into account whether this is plausible—there's a difference between plot twists and plot inconsistencies. In whichever case, the player must come away with an impression that there *is* some consistent backstory.

Besides the strongest constraint of consistency, which effectively creates a space of possible acceptable plots, we have a number of other features that encode a preference for some plots over others. For example, we'd like the backstory to be filled in gradually. If too many things are determined at once, it makes the story less dynamic; on the other hand, if things are revealed too slowly, the player will have no idea what's going on through most of the game. This balance is computed concretely by having an ideal arc for the degree to which the backstory is fixed at any given point; stories are rated based on how closely they match that arc.

In addition to the backstory arc, we have some preferences for how the player is led to suspect (but not be sure) of various bits of the backstory. One interesting feature is to prefer stories following a "Mobius rule", where every story has at least one major issue on which the player suspects something that turns out to be false (*i.e.* a plot twist).

Since there are some nice aspects to karmic systems, we have a component that essentially adds them back in at a global level. In particular, the backstory that's revealed ought to make sense in light of what the player does: If the player keeps acting in a fairly villainous manner, the revealed backstory shouldn't be a very positive one. This differs somewhat from karmic systems in that it's dealt with at the global level and traded off at that level against other evaluation components, rather than being maintained as a running number that's queried by individual rules.

One further issue is that the characters in the Neverwinter Nights engine are fairly "dumb": We have to manipulate them by changing specific bits of their dialog trees, rather than giving higher-level commands like "talk to the player". We mitigate this problem somewhat by enacting DM actions through an intermediate layer that takes care of some of the simpler refinements. For example, swapping out bits of dialog is no longer done at the DM level; instead, the DM makes more general decisions about what should be said. Similar refiners take care of other frequent operations like placing items. Ideally we'd still like more intelligent characters and a more full-fledged set of refiners, but even this relatively simple middleware layer has let us avoid many of the more tedious problems.

Conclusions

We've discussed applying declarative optimization-based drama management to designing a new game, *The Guilty*, that was infeasible to implement with traditional methods.

We found three primary benefits to using DODM. First, encoding authorial goals in an evaluation function rather than distributing them implicitly throughout the game code makes it easy to tweak goals during iterations of the design process, or even to completely change the feel of the game. Second, evaluation functions provide a natural way to express general global preferences, like preference for high spatial locality of actions. Finally, the fact that decisions are made at a global level with complex tradeoffs calculated by an automatic optimization process allows games to be modified (or written incrementally) without having to tediously re-balance the already-written portions each time.

Future Work

There are a number of other possible applications of DODM that would be interesting to explore. In particular, imbuing massively multiplayer online games with strong stories has been a longstanding problem in which we think DODM might prove useful. In addition, we've demonstrated one new type of game that becomes feasible only with DODM; there may well be other novel uses it enables.

There are several architectural and technical areas of possible future work as well. Our experiences suggest that having at least semi-intelligent refiners available to implement DM actions would be very useful to a game author. This could be accomplished by extended DODM to a hierarchical system of some sort, or by using it in a game in which the characters are written as agents that can be given higher-level commands to execute. In addition, DODM focuses entirely on plot as defined by plot-point sequences; it may be preferable to also address sub-plot issues such as the player getting stuck or off track. Finally, although it was not the focus of this paper, successfully running the optimization step currently requires a degree of familiarity with reinforcement learning; it would be nice to automate it fully so it can be reliably used as a black box.

References

Nelson, M. J.; Roberts, D. L.; Isbell, Jr., C. L.; and Mateas, M. 2006. Reinforcement learning for declarative optimization-based drama management. In *Proc. Fifth Intl. Joint Conf. on Autonomous Agents & Multiagent Systems*.

Weyhrauch, P. 1997. *Guiding Interactive Drama*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.