Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2025)

Adversarial Strong Story Experience Management

Valentina Genoese-Zerbi, Justus Robertson

School of Interactive Games and Media
Rochester Institute of Technology, Rochester, NY, USA
{vg1566 | jjrigm}@rit.edu

Abstract

In strong story experience management problems, an automated storytelling agent balances player autonomy with narrative structure in the context of an interactive story game world. However, it is possible for the game world to get softlocked in states outside narrative structures specified by the game designer. These states are called *dead-ends*. In this paper, we revisit *adversarial* strong story experience management, a framing of the experience management problem that models interactive storytelling as an adversarial game where dead-ends are losses. This framing is adversarial against narrative softlocks, not necessarily the player. We present a novel agent based on adversarial search and deep reinforcement learning, which is trained to avoid dead-ends while preserving player autonomy. We compare our approach to a reactive, narrative plan-based *mediation* system on a test set of games compatible with current narrative planning techniques. We show that our adversarial architecture outperforms narrative mediation on a suite of dead-end metrics during game trace and breadth-first tests of state transition system exploration, using classical and intentional planning domains.

Introduction

Experience management is the process of dynamically managing an interactive game environment to mediate between player autonomy and game designer control (Riedl and Bulitko 2013). This term is a generalization of *drama management*, the problem of directing a virtual, adaptive narrative where the player acts as one of the story characters. There are many philosophical and technological approaches to experience management, including sequential behaviors (Mateas and Stern 2002) and search-based drama management (Nelson and Mateas 2005). *Strong story* experience management is a particular approach that focuses on generating full story scripts for the NPCs to follow, in contrast to *emergent narrative* (Aylett 1999), which focuses on allowing narrative context to naturally emerge from interactions between the player and strongly-authored NPCs.

A leading approach for strong story experience management utilizes automated planners to author storylines used to control NPCs (Rivera et al. 2024). *Narrative planners* are used to add rich narrative structures to the stories generated (Rivera et al. 2024). Such structures include character intentionality, which prevents characters from taking actions that do not make progress towards their own intended goals (Riedl and Young 2010; Haslum 2012; Teutenberg and Porteous 2013; Ware and Young 2014). These storylines generated using narrative planners can be sequenced together using a process called *mediation* (Riedl, Saretto, and Young 2003) to create a tree of plots that branches to accommodate player actions while still solving the narrative plan. However, it is possible for the player to reach areas of the game world's state space that are mutually exclusive with the solutions to the planning problem. These areas are called dead-ends (Lipovetzky, Muise, and Geffner 2016) and they softlock the experience manager and player away from the story intended by the game designer. Research has made strides to address this problem, but the different approaches require an extra design burden (Harris and Young 2009; Mawhorter and Smith 2021), reasoning to track and exploit player knowledge (Robertson and Young 2018), or additional assumptions about time and player actions (Ware et al. 2022).

In this paper, we address the dead-end problem by returning to *adversarial search*, a technique used early in the history of drama management (Weyhrauch and Bates 1997). This is not necessarily an adversarial game against the player, but against narrative softlocks, which can be encountered by accident. We leverage advancements made in adversarial search using MCTS (Chaslot et al. 2008) and reinforcement learning (Madeira, Corruble, and Ramalho 2006) as combined in the AlphaZero framework (Silver et al. 2018) to create an adversarial experience management agent that trains through self-play. We modify the AlphaZero framework, similar to (Goldwaser and Thielscher 2020), to allow for cooperative and non-symmetric games in order to control experience management problems. We synthesize a set of dead-end formula (Lipovetzky, Muise, and Geffner 2016) from Planning Domain Definition Language (PDDL) (McDermott 2000) descriptions to identify loss states. And we compile (Haslum 2012) intentional narrative problems into classical PDDL, to be solved by our system.

Using this method, we train an experience manager model in a suite of classical and intentional planning game worlds and pit them against a traditional, reactive mediation agent (Robertson and Young 2014) in both a breadth-first and depth-oriented test with metrics related to the number of

dead-ends encountered. We find that our adversarial framework outperforms reactive mediation in nearly every test.

Related Work

We present a deep reinforcement learning-based strong story experience management framework that we benchmark against a reactive plan-based narrative mediation system. In this section, we provide backgrounds for each of these intersecting research areas.

Automated Planning and Dead-Ends

Planning is a well-established technique for automatically generating linear stories (Rivera et al. 2024) and can be modified to embed interesting narrative structures, like intentional character behavior (Riedl and Young 2010; Haslum 2012; Teutenberg and Porteous 2013; Ware and Young 2014). Modern planners use a series of pre-processing steps to efficiently search their state transition systems for states that satisfy goal conditions (Ware and Siler 2021). One of these pre-processing steps is dead-end identification. Similar to softlocks in games, dead-ends are states where there exists no possible path through the state transition machine to the goal. These states can be identified by finding collections of state literals that are mutually exclusive with goal conditions. Generating dead-end formulae is a non-trivial problem, but techniques exist that build the information from the related pre-processing steps of identifying *invariants* and *traps* (Lipovetzky, Muise, and Geffner 2016). The ability to quickly classify dead-ends is very important, because the AlphaZero framework needs to know when it has reached a losing state in order to provide negative reinforcement to its deep learning model.

Intentional Narrative Planning

When classical planners are used for narrative reasoning, it is possible for the planners to create low-quality stories in an effort to achieve the story goals as fast as possible (Riedl and Young 2010). There have been several developments by researchers to improve the quality of generated stories by embedding rich narrative structures, like character intention (Riedl and Young 2010; Haslum 2012; Teutenberg and Porteous 2013; Ware and Young 2014), narrative conflict (Ware et al. 2013), character regret (Martinelli and Robertson 2021), and character personality (Shirvani, Ware, and Baker 2022). One goal of our system is to be *domain-independent*. We want a process that works out of the box, without specialized narrative reasoning. To accomplish this goal, we utilize Haslum's narrative compiler (Haslum 2012) to transform intentional planning problems into classical PDDL descriptions, which are solvable by our system.

Narrative Mediation

When used to control NPCs in an experience management context, narrative plans must adapt and change based on the actions players take in the game environment. Narrative mediation (Riedl, Saretto, and Young 2003) is a process that analyzes story plans for logical inconsistencies a player can introduce. For each of these inconsistencies, the process of *accommodation* creates a contingency plan for the player action. Accommodation can be applied recursively to create a branching story tree, similar to a Choose Your Own Adventure (Riedl and Young 2006). The General Mediation Engine (GME) (Robertson and Young 2014) is one such narrative mediator, compatible with many different planners. Unfortunately, mediation does not actively steer players away from potential dead-ends. Prior work has addressed this problem (Harris and Young 2009), but requires an additional design burden for the author to embed contingencies in the game world mechanics. Our system is proactive through adversarial search and training. We use reactive mediation, as implemented by GME, to benchmark our proactive deep reinforcement learning experience management system.

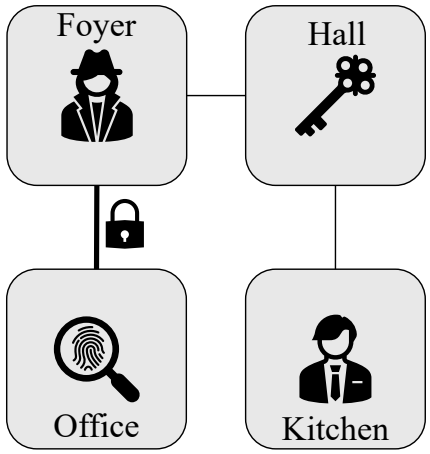

Figure 1: The initial state of the *Mansion* problem. The Player is at the Foyer, the Office Key is at the Hall, the Evidence is in the locked Office, and the NPC Butler is at the Kitchen. The goal is for the Player to acquire the Evidence.

Adversarial Search and AlphaZero

One of the first approaches to drama management was an adversarial search framework (Weyhrauch and Bates 1997). We return to this idea, but frame it to avoid narrative softlock that occurs at dead-ends within narrative planning problems. To do this, we leverage MCTS (Chaslot et al. 2008) and reinforcement learning (Madeira, Corruble, and Ramalho 2006) as implemented by AlphaZero (Silver et al. 2018), a generalization of the AlphaGo (Silver et al. 2017) system that learns classical board games entirely through self-play. AlphaZero makes a variety of assumptions about the games it plays, including requirements that they be zero-sum, symmetric, and use game boards with rotational and reflective symmetries. A symmetric game is one where all players have the same roles, including movesets, goals, and loss states. A board with rotational and reflective symmetry is one where the board can be rotated and flipped and still convey the same information. Our experience management problems are not strictly zero-sum, have asymmetric roles, and most interactive story worlds do not take place on symmetrical

```
(define (domain mansion)
   (:requirements :typing)
   (:types
      player npc - chr
      key ev - item
      chr item - thing
      thing loc - object)
   (:predicates
      (at ?thing - thing ?loc - loc)
      (has ?chr - chr ?item - item)
      (lock ?from - loc ?to - loc)
      (con ?from - loc ?to - loc))
   (:action move
      :parameters
         (?c - chr ?f - loc ?t - loc)
      :precondition
         (and
            (con ?f ?t)
            (at ?c ?f))
      :effect
         (and
            (not (at ?c ?f))
            (at ?c ?t)))
   (:action pickup
      :parameters
         (?c - chr ?i - item ?l - loc)
      :precondition
         (and
            (at ?c ?l)
            (at ?i ?l))
      :effect
         (and
            (not (at ?i ?l))
            (has ?c ?i)))
   ...)
```

Figure 2: A fragment of the PDDL Domain for the *Mansion* example. Not pictured is the final action template, *unlock*, which requires a key to remove a (lock ?from ?to) fluent and replace it with a corresponding (con ?from ?to), which makes the ?to location accessible if a character is at ?from.

boards like that of Chess or Go. Similar to (Goldwaser and Thielscher 2020), we relax these simplifying assumptions in our implementation and allow it to parse and manipulate state machines specified in PDDL.

Background

To begin, we describe a simple running example and use it to introduce modeling game worlds as formal state transition systems using the PDDL action language. PDDL is used to describe story world mechanics for narrative mediation systems and we use it as input to our experience manager.

Example Problem

We start by introducing an experience management problem set inside a mansion where the player must acquire and use items to make progress as a detective, like a murder mystery text-adventure game. The mansion has four rooms:

```
(define (problem 01)
   (:domain mansion)
   (:objects
      detective - player
      butler - npc
      foyer hall kitchen office - loc
      office_key - key
      evidence - ev)
   (:init
      (at detective foyer)
      (at office_key hall)
      (at butler kitchen)
      (at evidence office)
      (lock foyer office)
      (con office foyer)
      (con foyer office)
      (con hall foyer)
      (con foyer hall)
      (con hall kitchen)
      (con kitchen hall))
   (:goal (has detective evidence)))
```

Figure 3: A PDDL Problem for the *Mansion* domain. This corresponds to the diagram in Figure 1, described as a conjunction of state fluents in the *:init* list. The designer's goal is for the player to have the evidence locked in the Office. Either the player or NPC needs to open the Office.

the Foyer, Hall, Office, and Kitchen. The door between the Foyer and the Office is locked. A critical piece of evidence is in the Office. The key to the Office door is at the Hall. The Player is at the Foyer and the NPC Butler is at the Kitchen. The experience manager's goal is for the Player to enter the Office and collect the Evidence. We call this example problem *Mansion* and show a diagram of its initial state in Figure 1. We use this example to illustrate how story games are modeled with PDDL.

Action Language Encoding

Action languages are used to formally encode how agent actions affect a state transition system over time for artificial intelligence and robotics domains. PDDL (McDermott 2000) is the standardized action language used to encode planning problems for the International Planning Competition (Vallati et al. 2015). PDDL specifications consist of two components: the *domain* and *problem*. In our case, PDDL domains contain a type hierarchy for objects, the set of predicate relationships used to describe world states, and action templates that define what characters can do in the world and when. A description of the *Mansion* domain is given in Figure 2. PDDL problems contain the set of typed objects that exist in the world, the initial state description, and the goal equation. A description of the *Mansion* problem can be seen in Figure 3. PDDL states are comprised of *fluents*, built by binding world objects to predicates according to their types. For example, the predicate:

(at ?item - item ?loc - loc)

can be bound in eight possible ways, given the typed objects in the *Mansion* problem:

(at evidence foyer) (at evidence office)
(at evidence hall) (at evidence kitchen)
(at office_key foyer) (at office_key office)
(at office_key hall) (at office_key kitchen)

PDDL domains also specify action templates that describe how characters can act, when, and what effects their actions have on the world. Similar to predicates, these templates contain typed parameters that are bound to objects from the problem's world description according to the type hierarchy to create fully ground actions. These actions contain a set of preconditions that must match the current state fluents in order for the action to be performed. When an action is taken, it creates a successor state by modifying the fluents of the original state according to the atoms in the actions effect set.

Together, the initial state and set of fully ground actions create a state transition system that can be represented as a directed graph with nodes corresponding to possible state configurations and edges representing enabled character actions. A solution to the planning problem is a path through this state transition system from the initial state to a state matching the goal conditions specified by the PDDL problem. Our experience manager uses PDDL to define its game mechanics and its *win* condition is specified by the PDDL problem's goal. However, a gameplaying framework needs to know in what states it has *lost* in addition to the states where it has won. To specify these states, we find the PDDL problem's *dead-ends*.

Dead-End Identification

We use a pre-processing algorithm designed by Lipovetzky et al. (Lipovetzky, Muise, and Geffner 2016) to identify dead-end formulae before plan search begins by first identifying invariants and traps:

Definition 1 *An invariant is a formula that is true in all reachable states of the state transition system.*

An invariant fluent in *Mansion* is (con kitchen hall). No matter how many times the characters move, pickup items, or unlock doors, the connection between the kitchen and hall will always exists. A trap is related to an invariant:

Definition 2 *A trap is a formula such that once a state satisfies the formula, it is invariant in successor states.*

Traps are formulae that, once satisfied, are invariant within the reachable subgraph from the state that satisfies them. For example, in our *Mansion* problem, (has detective office_key) begins *False*, which means the singleton formula is not satisfied by the initial state. However, if the Detective picks up the Office Key, they have no way to drop it. So, (has detective office_key) is a trap, persisting once satisfied.

Building on this definition, a dead-end is a trap where at least one component of its formula is mutually exclusive with a goal condition. If the trap is sprung, the state subgraph entered is mutually exclusive with one of the designer's narrative goals. Lipovetsky et al.'s algorithm detects traps by building a tree of partial world states navigated by actions. If a part of the tree is found where the prior states cannot be navigated through any set of actions, a trap has been found. If the partial state is mutually exclusive with the goal state, the trap is also a dead-end. Using the list of dead-end formula, we identify loss states for our experience manager.

Intentional Planning Encoding

Additional logical operators are needed when encoding intentional planning problems. In addition to standard parameters, preconditions, and effects, actions must indicate what characters are required to consent to the action and what intentions are added or removed. Instead of building a planner with special reasoning for this extension, Haslum presents multiple compilation techniques that compile the additional intentional planning information into a classical PDDL representation (Haslum 2012). In this paper, we use meta-planning compilation, which allows characters to adopt intentions that meet preconditions of actions that cause their current intentions to be met. In this approach, a direct action can be taken if the acting character has an intention that is fulfilled by the action. The intentions themselves, rather than being nested atoms, become combined into one single atom. For example:

(intends butler (has butler key))

becomes

(intends-has butler butler key)

where the name of the atom is attached to the *intends* label and the first bound object indicates who holds the intention.

Adversarial Strong Story Experience Management

Given a PDDL state transition system, a set of goal conditions, and a set of dead-end formula, we define the adversarial strong story experience management gameplaying problem as an adversarial game that begins at the initial state, the player and experience manager take turns acting, the experience manager *wins* by reaching a goal state, and *loses* by reaching a dead-end. Note that the term *adversarial* in this context refers to the experience manager actively avoiding dead-ends, not necessarily opposing the player. An adversarial experience manager will only be locked into a zero-sum game with the player if the player is working directly against the designer's intent and attempting to accomplish a dead-end. This will usually not be the case, and often the experience manager and player will behave cooperatively to achieve story objectives. Most dead-ends are not arrived at intentionally, and this adversarial approach safeguards the player and system against them. Given this problem definition, we present a deep reinforcement learning framework, modified from AlphaZero, that solves these adversarial strong story experience management problems.

Experience Management Framework

An illustration of the system architecture is given in Figure 4. The game world and goal are specified in PDDL and passed to Lipovetzky et al.'s dead-end detection algorithm (Lipovetzky, Muise, and Geffner 2016) to generate a list of

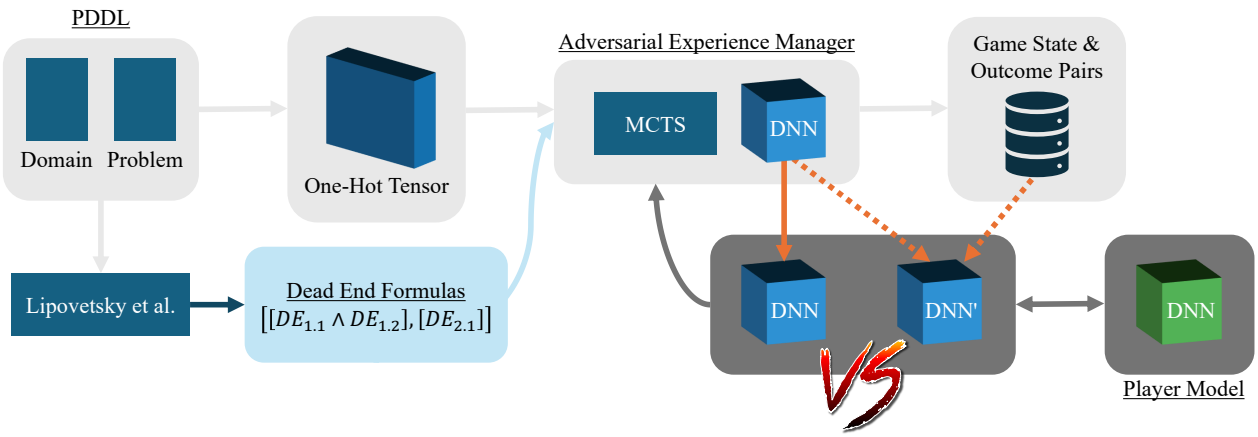

Figure 4: Illustration of the system architecture. The process begins with a formal description of the story game world's state transition system and author goal, formulated in PDDL. The PDDL is used to create a playable game in Python and further funneled in two directions: it is processed to find dead-end formula (Lipovetzky, Muise, and Geffner 2016) that identify soft-lock "loss" states and translated into a one-hot encoded state tensor as input to the deep reinforcement learning module. The reinforcement learning module follows the AlphaZero (Silver et al. 2018) pattern of data generation, learning, then competition to decide which model moves to the next round. To account for asymmetry, a separate model is trained to act as the player.

dead-end formula for that game. This is passed into a modified version of the AlphaZero adversarial gameplay framework that combines Monte Carlo Tree Search (MCTS) with deep reinforcement learning and self-play.

AlphaZero Modifications

Unfortunately, AlphaZero (AZ) does not work off-the-shelf with our experience management problems. It makes several simplifying assumptions that do not hold in most of our domains, two of the most important being:

- **Zero-Sum Games** - AZ assumes that anything good for one player is proportionally bad for the other. However, most experience management domains are at least partially cooperative between the player and storyteller. Despite the *adversarial* title, this system is adversarial against dead-ends, which are not usually what the player is attempting to accomplish in the game environment. It therefore follows that most problems are not zero-sum between the experience manager and player.
- **Asymmetric Games** - AZ also assumes that the possible actions and payoffs are mirrored between both players, like in a game of Chess. In our case, the player may control a character with very different possible actions than the NPCs controlled by the experience manager. If the player's internal goal is to achieve dead-ends, our system behaves in a traditionally adversarial manner.

Existing work has explored how to modify AZ to create a deep reinforcement learning framework for general game playing (Goldwaser and Thielscher 2020) and we similarly took measures to relax these two AZ assumptions. To support non-zero sum gameplay, we modified MCTS to use separate state-value estimations for each player. Once a set of game results or estimated utility values is retrieved for both players for a game state, both values are back-propagated such that the value for the appropriate player perspective is stored in the tree at each corresponding level. To support asymmetrical gameplay, the reinforcement learning model training is split in two. Performance analysis within one iteration was changed from the new and current versions of a single DNN model playing directly to two different models playing against the current version of the other. The new deep-learning cycle follows these steps:

Step 1.) The current experience manager (EM) and player (P) models play games against each other using MCTS coupled with their DNNs. Data of each state, decision, and game outcome for each game played is collected and used to train two new models: EM′ and P′.

Step 2.) The current (EM) and newly trained (EM′) models for the experience manager both play games against the current player (P) model. If EM′ wins more than 10% more against the P model compared to EM, EM′ becomes the current model. Otherwise, EM′ is discarded.

Step 3.) These steps are repeated for P, with P and P′ playing against EM. This process repeats until desired performance is reached.

In addition to wins and losses, the models can *tie* when the game reaches a depth-limit without arriving at a goal or dead-end. We count these ties as wins for the EM when performing model comparisons. The objective of adversarial experience management is to reach a goal state, which is possible as long as it has avoided a dead-end. Any state that is still on some possible path to a goal state is valuable. This is particularly true in domains where the experience manager cannot force a goal state without the player's help.

Evaluation

In this section, we benchmark our Adversarial Experience Management (AEM) approach against traditional plan-

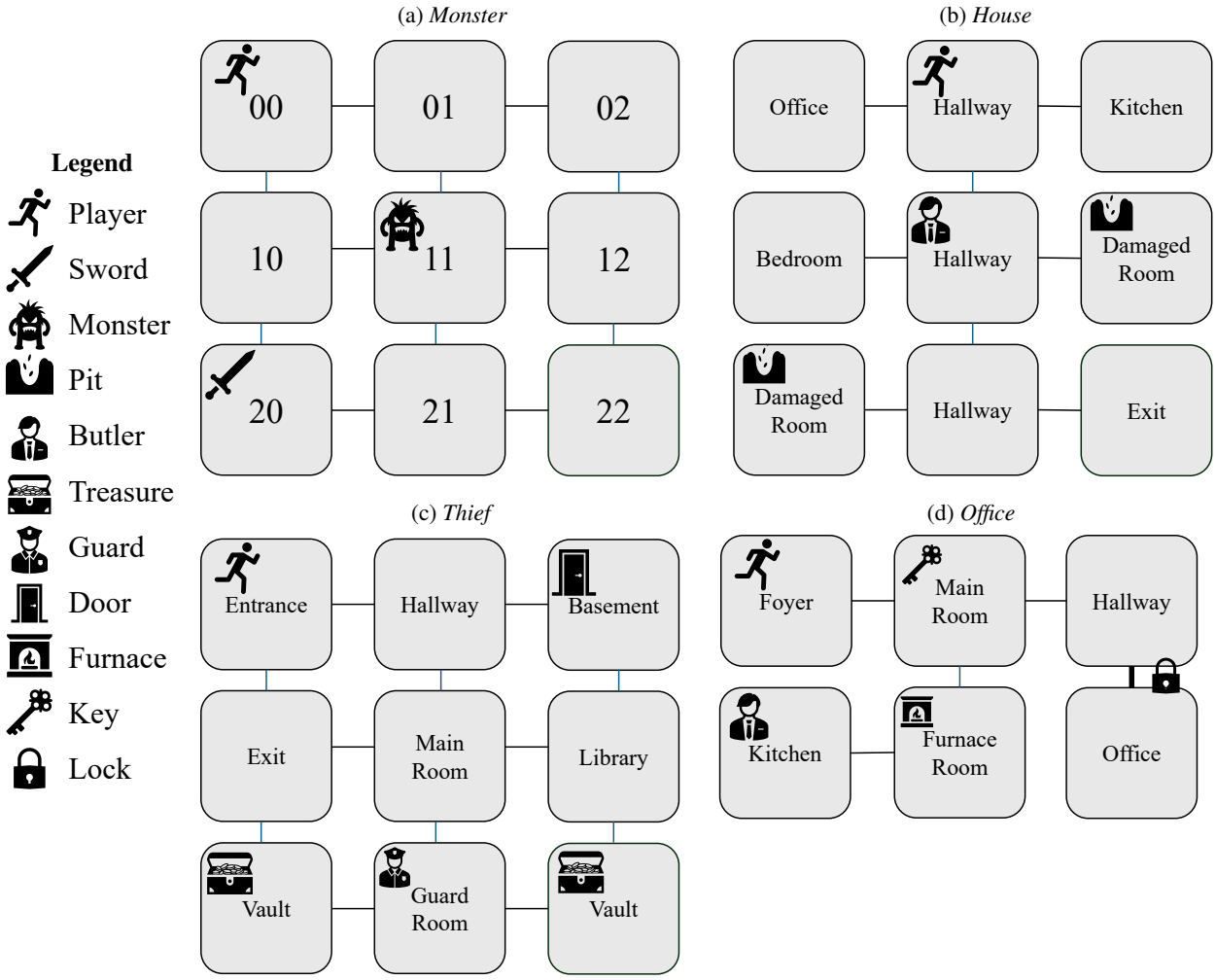

Figure 5: The four example worlds used to benchmark Adversarial Experience Management against the General Mediation Engine: *Monster*, *Office*, *Thief*, and *House*. Each features a single player character and one NPC controlled by the system.

based mediation, as implemented by the General Mediation Engine (GME) (Robertson and Young 2014), using Fast Downward (Helmert 2006) to generate linear solution plans. We present a set of six example problems for the two systems to compete on, each with different properties that impact experience management ability. The first five example problems are classical PDDL and the final problem is an intentional planning domain, as compiled by Haslum's meta-planning approach (Haslum 2012). We compare AEM against GME's performance on each of these benchmarks.

We run two types of tests: 1.) a breadth-first search (BFS) expansion of the state transition system in every direction the player could choose, allowing the experience manager to respond at each level; and 2.) a series of 100 depth-based single game traces using a player model that takes random actions. The five classical problems are narrative-style text adventures encoded in PDDL. The intentional problem contains character intentions and was compiled to a max intention depth of 4. We run BFS to a depth of 8 or 6, depending on the complexity of the game world. For the game trace test, we stop at a maximum depth of 50 turns and treat any game that reaches this depth as a draw. We reward AEM with $+1$ for reaching a goal state, -1 for reaching a dead-end, and $+0.5$ for a draw. We reward $+0.5$ instead of 0 for a tie to make AEM risk-averse, as a tie outcome means that reaching a goal is still possible.

AEM is trained and runs on a dedicated PC using Ubuntu 22.04 with an NVIDIA GeForce RTX 3090 and 63G of RAM. We use the same computer to run GME, but don't collect its performance statistics beyond its decisions. Each AEM model is trained until further iterations show no significant improvements.

Classical Problems

In this section, we present a set of five experience management problems modeled with PDDL using classical planning. The five problems increase in complexity and are grouped in three stages: easy, intermediate, and complex.

	Dead-Ends at Depth								
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***DE%***
Office									
GME	0	0	0	1	5	25	105	421	2.5%
AEM	0	0	0	0	0	0	0	0	0%
Monster									
GME	0	0	3	12	48	174	633	2268	8.8%
AEM	0	0	0	0	0	0	0	0	0%
Thief									
GME	0	1	3	12	40	147	-	-	7.10%
AEM	0	1	1	1	1	1	-	-	0.06%
House									
GME	0	1	4	15	50	161	-	-	14.40%
AEM	0	0	1	1	1	1	-	-	0.10%

Table 1: Results of breadth-first state transition system expansion following AEM and GME directions. *Office* and *Monster* are expanded to a depth of 8, while *Thief* and *House* to depth 6 due to higher complexity state spaces. The final column reports the percent of total tree branches that result in a dead-end at the final calculated level. Cells marked with a colored background indicate top performance in that category. AEM strictly outperforms GME past depth 3 and on the final percentage results.

The full results of the breadth-first test are shown in Table 1 and the full game trace results are shown in Table 2. The first two domains, Monster and Office, have low state complexity and allow experience managers a strong amount of control over game outcomes:

Problem 1: Monster

- *Description*: The *Monster* domain is shown in Figure 5a. This domain is the most game-like, and is played on a 3x3 open grid that contains a sword, a monster, and an end space (2,2). The goal is for the player to kill the monster and then go to the end space. If the player is ever co-located with the monster without the sword, the monster will eat the player. If the player is co-located with the monster while holding the sword, the monster will be slain. The dead-end in this domain is the player's death, as there is no method to re-spawn once dead.
- *Training Statistics*: AEM was trained for 3 iterations that took 3 hours, 5 minutes, and 16 seconds to complete.

Problem 2: Office

- *Description*: The *Office* domain is shown in Figure 5d. This problem is set in a house with 6 rooms that contain an office with a locked door, the key to the office, an NPC, and a furnace. The goal is for the player to enter the office. If the player has the key and is in the same room as an enabled furnace, they can destroy the key. The NPC can move around and permanently disable the furnace. The dead-end in this domain is the key being destroyed before the office door is unlocked.
- *Training Statistics*: AEM was trained for 1 iteration that took 1 hour, 8 minutes, and 51 seconds to complete.

Results

- *Data*: AEM strictly outperforms GME on all breadth and depth-based metrics. 0% of AEM branches result in a dead-end in the first 8 levels of the BFS tree, compared to 2.5% for GME on *Office* and 8.8% on *Monster*. On the depth-based test, more AEM game traces result in goal states for both *Office*, 54% to 34%, and *Monster* (just barely), 13% to 11%. There were large differences in the percentage of play-outs that resulted in dead-ends as well as final outcome valuations, all in AEM's favor.
- *Discussion*: In both *Office* and *Monster*, it is possible for the experience manager to entirely avoid dead-ends. In *Office*, the optimal strategy is to move directly to the furnace and disable it. In *Monster*, the best moves for the monster will vary based on the position of the player and whether or not they have collected the sword. The monster can avoid combat until the player is ready. The results collected in both tests demonstrate AEM has learned optimal play, while GME is focused on carrying out plans rather than actively avoiding dead-ends.

The second two games, *Thief* and *House*, were designed to have larger state spaces and such that the experience manager cannot always avoid dead-ends through optimal play.

Problem 3: Thief

- *Description*: The *Thief* domain is shown in Figure 5c. The problem is set in a manor with 9 rooms that contain two treasures, a guard, a basement, and an exit. The goal is for the player to get to the exit with both treasures. The basement has a self-locking door, which will lock behind the player if they enter. The guard cannot enter the basement. If the guard moves to a room that the player is in, they will send the player back to the start.
- *Training Statistics*: AEM was trained for 10 iterations that took a total of 12 hours, 16 minutes, and 18 seconds to complete.

Problem 4: House

- *Description*: The *House* domain is shown in Figure 5b. The problem is set in part of a house consisting of 6 rooms and a hallway in three sections, each section connecting to two rooms. Two of the rooms have pits that

	Avg. Depth	*Goal %*	*DE %*	VAL
Office				
GME	32.73	34%	36%	13
AEM	37.39	54%	0%	77
Monster				
GME	17.06	11%	81%	-66
AEM	47.36	13%	0%	56.5
Thief				
GME	18.74	9%	87%	-76
AEM	43.95	8%	11%	37.5
House				
GME	16.68	17%	75%	-54
AEM	27.53	76%	2%	85
Apprentice				
GME	35.97	18%	32%	11
AEM	40.30	32%	15%	43.5

Table 2: Results of full game traces from initial state to goal, dead-end, or depth limit. This table shows the average depth, percentage goal outcomes, percentage dead-end outcomes, and a *value* metric. The value metric multiplies the proportion of win, loss, and ties by the reward values AEM is assigned for each outcome (+1, -1, +0.5) and sums the results. Cells marked with a colored background indicate top performance in that category. AEM outperforms GME across all categories, aside from a 1% difference in *Thief* goals.

characters can fall into and be unable to escape from if they enter the room. The goal is for the player to get to the exit room. The player can move from room to room. There is an NPC that can move from room to room and lock the door of a room they are adjacent to, preventing both the player and the NPC from entering.

- *Training Statistics*: AEM was trained for 5 iterations that took a total of 10 hours, 39 minutes, and 42 seconds to complete.

Results

- *Data*: AEM outperforms GME on all but one metric. Only 0.06% of AEM branches result in a dead-end in the first 6 levels of the BFS tree, compared to 7.10% for GME on *Thief* and 0.10% vs. 14.40% on *House*. On the depth-based test, more AEM game traces result in goal states for both *House*, 76% to 17%, but AEM underperforms GME by 1% on *Thief*, 8% to 9%. Again, there were large differences in the percentage of play-outs that resulted in dead-ends and final value metrics, with AEM leading in each.
- *Discussion*: In the *Thief* and *House* games, it is not possible for the experience manager to avoid dead-ends through optimal play. In *Thief*, the player can get to the basement before the experience manager can stop them. The results collected in DFS show the AEM learns to herd the player away from dead-ends that GME does not.

A final classical domain is included as a fifth entry into the depth-test, shown in Table 2. In this Apprentice problem, two thieves must work together to disarm a series of traps in a mansion to steal a heavily guarded gem and escape. AEM strictly outperforms GME on this larger domain, which has the highest average depth of any problem tested for GME. AEM finds a goal state in 32% of play traces compared to 18% from GME, and AEM hits a dead-end in only 15% of games compared to 32% from GME.

Princess				
	Avg. Depth	*Goal %*	*DE %*	VAL
GME	30.99	18%	49%	-14.5
AEM	44.86	14%	6%	48

Table 3: Full game trace results for the intentional *Princess* domain. GME finds 4% more goals, but reaches 43% more dead-ends than AEM. This difference is reflected in the value metric, with AEM outperforming by a margin of 62.5.

Intentional Problem

The intentional domain *Princess*, shown in figure 6, features a hero with a crown that must be returned to a princess who is locked inside a tower. The initial problem contained 8 possible intentional actions and 8 identical non-intentional player-only actions. This was done to allow the player to act without character intention tracking and restrictions, so they are free to act as they please. There was additionally 1 motivation action that gives a character an intention. Post-compilation, the problem contains a total of 32 possible actions. It additionally features 5 locations and 2 NPCs:

- *The Locksmith* - Runs a locksmithing shop. Because a bakery nearby has caught fire and spread to her shop, she intends to extinguish both fires. In her shop, there is a lockpick that the player can steal if she is not around, or buy if she is and the player has money.
- *The Princess* - Trapped in a tower. She intends to be in the forest and to have her crown. The princess can also leave if the door is unlocked, which the player can achieve by picking the lock with a lockpick or by unscrewing the door with a screwdriver from a nearby house.

If the player destroys the crown in a fire, a dead-end is reached. AEM was trained for 3 iterations that took a total of 6 hours, 5 minutes, and 1 second to complete.

Results

- *Data*: The results are shown in Table 3. AEM outperforms GME on the average depth reached, dead-end percentage, and total value, while GME reached a goal state in 4% more outcomes than AEM.
- *Discussion*: Although GME reaches slightly more goals, AEM performs much better overall according to the value metric due to reaching 43% less dead-ends. This means that a much higher proportion of AEM's games are ending in ties where the goal has not been reached but is still accessible. GME is playing greedily, which gives it a slightly higher win percentage but an overall net negative reward given our valuation of outcomes (+1 for wins, +0.5 for ties, and -1 for losses).

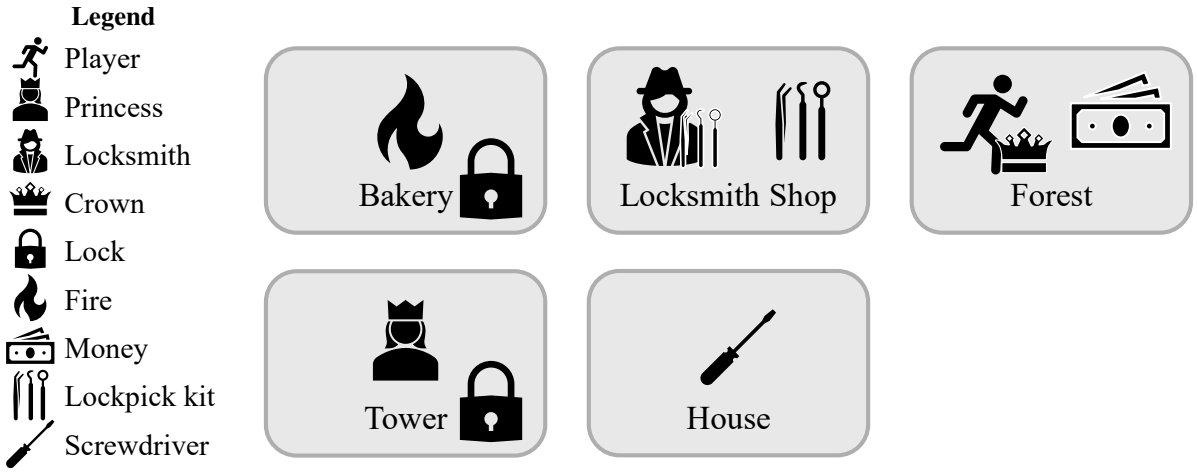

Figure 6: The intentional *Princess* domain.

The Mann Whitney U test was performed comparing dead ends and goal states achieved in the depth-first search across all domains. AEM was found to achieve dead ends less often ($p<0.001$) and goals more often ($p<0.001$) than GME.

Discussion

In nearly all of our tests, AEM matched or exceeded narrative mediation in the proportion of goals found while dominating GME in terms of dead-end encounters. The only outliers are the goal proportions of *Thief* and *Princess*, but the margins were small at a 1% and 4% difference. This is the expected result, as GME reactively traverses its story tree based on the actions of the player whereas AEM actively safeguards against dead-end states it trains to avoid. GME may eagerly reach more goals in some cases, but AEM's careful approach keeps the experience manager away from dead-ends. This is reflected by the value metric across all problems, include *Thief* and *Princess*, where AEM wins by large margins despite reaching fewer overall goals.

The AEM framework does require offline training in order to control the story world, but once it has converged on a policy it can run in real-time with relatively little search compared to GME, which must either pre-cache full story trees or compute plans on the fly. Overall, our results show that the AEM framework can be used to control text-adventure-style experience management domains, including one modeled with intentional planning, at a higher level than GME with an investment of 1-13 hours of GPU training time.

Limitations and Future Work

This work can be extended both in terms of the types of narratives that can be produced as well as the fidelity of the game worlds it can manage. First, there is a deep well of narrative planning techniques beyond character intentionality that AEM currently makes no use of, including narrative conflict (Ware et al. 2013), character regret (Martinelli and Robertson 2021), and character personality (Shirvani, Ware, and Baker 2022). To use these out-of-the-box, narrative compilation would need to be extended to model other areas of narrative reasoning beyond intentionality. A step was taken in this direction by Christensen et al.'s intentional planning compiler that allows for character beliefs and failed actions (Christensen, Nelson, and Cardona-Rivera 2020).

Additionally, the types of game worlds this architecture controls is limited to text-adventure style worlds with discrete turn-taking. However, advancements have been made in the world of adversarial gameplaying to extend the deep reinforcement learning, self-play architecture from board game to strategy game environments, like *StarCraft II* (Vinyals et al. 2019). Using similar techniques could help relax current constraints on the fidelity and size of the text-based game worlds the current architecture can manage.

Conclusion

We present adversarial strong story experience management (AEM), an adversarial gameplaying approach to experience management that prioritizes avoiding dead-ends, which are regions in the state space where the player gets soft-locked away from the designer's intended narrative structure. We described the technical implementation of our system and provided an evaluation where the implementation was benchmarked against a traditional narrative mediation system. We found that AEM outperformed traditional mediation on a number of metrics and tests related to dead-end encounters. We conclude that AEM is a powerful approach in the field of experience management and further work is warranted to tune AEM systems to orchestrate experiences with narrative properties aside from intentional character behavior and games larger than text-based adventures.

Acknowledgments

This work was supported by the National Science Foundation under Grant No. #2303650. We would also like to thank our anonymous reviewers, who offered helpful feedback during the peer review process.

References

Aylett, R. 1999. Narrative in virtual environments-towards emergent narrative. In *Proceedings of the AAAI fall symposium on narrative intelligence*, 83–86. USA.

Chaslot, G.; Bakkes, S.; Szita, I.; and Spronck, P. 2008. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, 216–217.

Christensen, M.; Nelson, J.; and Cardona-Rivera, R. 2020. Using domain compilation to add belief to narrative planners. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, 38–44.

Goldwaser, A.; and Thielscher, M. 2020. Deep reinforcement learning for general game playing. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 1701–1708.

Harris, J.; and Young, R. M. 2009. Proactive mediation in plan-based narrative environments. *IEEE transactions on computational intelligence and AI in games*, 1(3): 233–244.

Haslum, P. 2012. Narrative Planning: Compilations to Classical Planning. *Journal of Artificial Intelligence Research*, 44: 383–395.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Lipovetzky, N.; Muise, C.; and Geffner, H. 2016. Traps, Invariants, and Dead-Ends. *Proceedings of the International Conference on Automated Planning and Scheduling*, 26(1): 211–215.

Madeira, C.; Corruble, V.; and Ramalho, G. 2006. Designing a reinforcement learning-based adaptive AI for large-scale strategy games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 2, 121–123.

Martinelli, M.; and Robertson, J. 2021. A Plan-Based Formal Model of Character Regret. In *Interactive Storytelling: 14th International Conference on Interactive Digital Storytelling, ICIDS 2021, Tallinn, Estonia, December 7–10, 2021, Proceedings 14*, 107–117. Springer.

Mateas, M.; and Stern, A. 2002. A behavior language for story-based believable agents. *IEEE Intelligent Systems*, 17(4): 39–47.

Mawhorter, R.; and Smith, A. 2021. Softlock detection for super metroid with computation tree logic. In *Proceedings of the 16th International Conference on the Foundations of Digital Games*, 1–10.

McDermott, D. M. 2000. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2): 35.

Nelson, M.; and Mateas, M. 2005. Search-based drama management in the interactive fiction Anchorhead. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 1, 99–104.

Riedl, M.; Saretto, C. J.; and Young, R. M. 2003. Managing interaction between users and agents in a multi-agent storytelling environment. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, 741–748. New York, NY, USA: Association for Computing Machinery. ISBN 1581136838.

Riedl, M. O.; and Bulitko, V. 2013. Interactive narrative: An intelligent systems approach. *Ai Magazine*, 34(1): 67–67.

Riedl, M. O.; and Young, R. M. 2006. From linear story generation to branching story graphs. *IEEE Computer Graphics and Applications*, 26(3): 23–31.

Riedl, M. O.; and Young, R. M. 2010. Narrative Planning: Balancing Plot and Character. *Journal of Artificial Intelligence Research*, 39: 217–268.

Rivera, R. E. C.; Jhala, A.; Porteous, J.; and Young, R. M. 2024. The Story So Far on Narrative Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 489–499.

Robertson, J.; and Young, R. 2014. The general mediation engine. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 10, 65–66.

Robertson, J.; and Young, R. M. 2018. Perceptual Experience Management. *IEEE Transactions on Games*, 11(1): 15–24.

Shirvani, A.; Ware, S. G.; and Baker, L. J. 2022. Personality and emotion in strong-story narrative planning. *IEEE Transactions on Games*, 15(4): 669–682.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of Go without human knowledge. *Nature*, 550(7676): 354–359,359A–359L. Copyright - Copyright Nature Publishing Group Oct 19, 2017; Last updated - 2024-03-22.

Teutenberg, J.; and Porteous, J. 2013. Efficient intent-based narrative generation using multiple planning agents. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 603–610.

Vallati, M.; Chrpa, L.; Grześ, M.; McCluskey, T. L.; Roberts, M.; Sanner, S.; et al. 2015. The 2014 international planning competition: Progress and trends. *Ai Magazine*, 36(3): 90–98.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *nature*, 575(7782): 350–354.

Ware, S.; Garcia, E. T.; Fisher, M.; Shirvani, A.; and Farrell, R. 2022. Multiagent narrative experience management as story graph pruning. *IEEE Transactions on Games*, 15(3): 378–387.

Ware, S.; and Young, R. M. 2014. Glaive: a state-space narrative planner supporting intentionality and conflict. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 10, 80–86.

Ware, S. G.; and Siler, C. 2021. Sabre: A narrative planner supporting intention and deep theory of mind. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 17, 99–106.

Ware, S. G.; Young, R. M.; Harrison, B.; and Roberts, D. L. 2013. A computational model of plan-based narrative conflict at the fabula level. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3): 271–288.

Weyhrauch, P.; and Bates, J. 1997. *Guiding interactive drama*. Carnegie Mellon University Pittsburgh.