Perceptual Experience Management

Justus Robertson[1] and R. Michael Young[2], *Senior Member, IEEE*
[1]Department of Computer Science, North Carolina State University, Raleigh, NC 27695 USA
[2]School of Computing, University of Utah, Salt Lake City, UT 84112 USA

In automatically generated interactive narratives with strong story structure there is often a tension between player choice and author constraints. This tension arises when the player contradicts a story the system is in the process of telling. When this happens there are two options available to an interactive narrative storyteller that preserve the author's intentions. The first, called *accommodation*, finds a new story that matches the player's action while preserving author constraints. The second, called *intervention*, removes unwanted effects from the player's action. Neither of these alternatives are ideal. For accommodation, a new story is not always available. For intervention, the player could notice the inconsistent effects of their actions. In this paper we present a new approach to mediating between player choice and authorial constraints, called *perceptual experience management*, which mitigates drawbacks of both accommodation and intervention by incorporating a model of player knowledge into the strong story experience management process. This model is used to widen the space of possible accommodations and constrain the space of possible interventions to better balance author control and player choice. These two strategies are implemented in the General Mediation Engine, an experience manager capable of procedurally generating and revising a game world.

***Index Terms*—Interactive Narrative, Procedural Content Generation, Narrative Generation, Experience Management**

I. Introduction

NARRATIVE is one way humans structure and understand the world around us. One way to represent narrative is as a series of events, performed by story characters, that plays out in a story world. An interactive narrative (IN) is a story whose events are influenced by an outside participant able to manipulate world states or story characters. In automated strong story IN systems [1], a central artificial agent generates and adapts an interactive story around the actions of an outside participant. When the participant controls a story character in these systems, they may act contrary to the actions prepared for the character in the script generated by the author agent. This tension between player choice and authorial control is called the *the boundary problem* [2]. One way to overcome the boundary problem is to create a branching story that accounts for each sequence of actions the player could take.

Creating one series of interesting narrative events is a hard problem that requires time to plan and execute. The amount of material an author must generate to create an interesting story is called the *authorial burden*. One way to model a branching story is as a tree of overlapping linear stories that branches for each decision the player can make [3]. Each unique event sequence in the branching story, represented as a path through its tree data structure, increases its authorial burden. If an interactive narrative tree branches uniformly by two unique choices, then the tree's authorial burden will increase exponentially by a power of two for each choice layer it contains. This exponential increase in content for authoring interactive narratives limits the size and scope of story experiences human authors can create on their own. One way to mitigate the branching story authoring burden is to automatically create interactive narrative trees.

A process called *planning* can automatically construct linear stories with interesting narrative properties, like intentional character actions [4] and conflict [5]. The planning process takes as input a planning problem and produces a solution plan. A planning problem describes what exists in a world, how the world functions when agents take action, and how the world should be configured at the end of the plan. A solution plan is any series of actions taken by agents in the world that begins from the problem's initial configuration and produces a goal state. Through planning, the authorial burden of story generation can be reduced from authoring sequences of events to defining a story world and sets of goal conditions.

A process called *mediation* [6] generates branching IN trees through two subprocesses: *accommodation* and *intervention*. Mediation begins with a planning problem and produces an initial solution that serves as a path through the IN tree. Mediation then analyzes the plan to find every course of action the participant could take that would prevent the goal state from being achieved and creates a new planning problem for each action. Each solution plan serves as a new branch of the tree should the player decide to take the alternate action. This process is recursively invoked until no alternate course of action remains. This process, called *accommodation*, expands the branches of IN trees. One drawback to this solution is that not all planning problems can be solved and the participant could follow branches where no valid storyline exists.

An alternative to accommodation, called *intervention*, prevents new IN tree branches from being formed. It does this by temporarily changing how a participant's action transforms the story world such that it no longer contradicts the author's goals and does not break the current story. This method is useful when the player takes an action that cannot be accommodated. One drawback to intervention is that the participant may notice the world responds differently to their actions depending on the situation. This realization may break the user's immersion by making apparent a storyteller is at work behind the scenes changing game rules at the expense of player agency.

In this paper, we present a method called *perceptual experi-*

Corresponding author: Justus Robertson (email: jjrobert@ncsu.edu)

ence management that better mediates between player choice and author control by increasing the number of branches accommodation produces by widening the number of solution plans available to the planner. This new class of branches are generated by changing past story events while ensuring modifications are consistent with the player's experience and removing interventions that contradict the player's experience. These new branches allow the player to take more choices that are consistent with author constraints. These capabilities are implemented in a game engine that procedurally generates its interface based on the configuration of the underlying declarative story world state. This procedural content generation (PCG) pipeline allows the mediator to dynamically revise world states and mechanics to benefit author goals without the player realizing the story world has changed.

II. Related Work

The related work is divided into four sections: planning and narrative generation, interactive narrative generation, alibi generation, and procedural content generation.

A. Planning and Narrative Generation

One way to view narrative is as a series of logically and chronologically connected events performed by story characters and presented to an audience through some medium [7]. Planning is one way stories can be represented and generated [8]. A standard way of modeling planning problems is with with an action language called PDDL [9] (Planning Domain Description Language). One problem with planning as narrative generation is not all sequences of events are necessarily interesting stories. One open research question is how to reason about or identify interesting stories. Intentional planning [4], [10], [11] adds narrative reasoning to the planning process by expanding its PDDL input to include subjective goals for individual characters. These goals allow characters to only act when it is according to their interests. This additional layer of reasoning allows the algorithm to exclude stories where characters behave erratically or irrationally. Another algorithm, CPOCL [12], is an iteration on intentional planning that allows and identifies conflict between characters in story plans. A planner called Glaive [5] is a state space planner based on Fast-Forward [13] that produces plans equivalent to CPOCL. Our framework finds linear stories with Glaive as well as non-narrative planners like Fast Downward [14].

B. Interactive Narrative Generation

Approaches to generating interactive narratives can be classified along three dimensions: virtual characters, authorial intent, and player modeling [1]. These three dimensions classify systems based on how much freedom character agents have to act on their own versus how much control a central storyline has, whether the story is generated by an algorithm, and to what extent the storyline adapts to individual players.

One popular form of interactive storytelling is the Choose Your Own Adventure [15] (CYOA) book series. At the bottom of each CYOA page the book prompts the reader with a decision where each choice option corresponds to a page number. CYOAs are strongly authored and provide no character autonomy. On the other side of the spectrum from CYOAs, emergent narrative [16] systems, like FearNot! [17], take an automated, decentralized approach to IN. Emergent systems have high character autonomy and high automation. These systems tell stories through unscripted interactions between a user and autonomous virtual characters.

Between CYOAs and emergent narratives are systems like Façade [18]. Façade is a hybrid system that balances plot control with autonomous agents and authored content with automation. In Façade, the player assumes the role of a dinner guest and interacts with a virtual couple as they work through marital problems. The game's character behaviors are written with a reactive planning system, ABL, which is an iteration on the agent behavior architecture pioneered by the OZ Project [19] that organizes NPC behavior into story beats.

Our framework controls interaction using a central automatically generated script. The framework descends from Mimesis [20], a narrative control system integrated with the Unreal Tournament (UT) engine. Mimesis creates plans that correspond to actions virtual characters take in the UT environment. Mimesis uses mediation [6] to adapt its plan to player actions in the game. The data structures mediation creates to adapt its plan, called mediation trees, are representationally equivalent to a CYOA [3], but consist of stories that are automatically constructed with a planner.

Other plan-based systems have been created since Mimesis. ASD [21] is a mediation framework that decouples itself from any one game engine with an API that allows its mediator to sense an external game environment [22]. ASD moves closer to Façade's mixed story and character autonomy system by decoupling character control from the current plan. Another system, The Merchant of Venice [23], allows players to make interventions that change how characters behave. Finally, the PAST [24] system is built on the ASD framework and uses real-time planning to select between stories on the fly based on a predicted player type of the interactor.

C. Alibi Generation

Our framework increases the branching factor of mediation trees by taking advantage of their incomplete knowledge of the story world to create alibis. Alibis were first used by Sunshine-Hill [25] to make NPCs in large, sandbox worlds appear intelligent for relatively little computational cost. The system controls NPC actions according to a random distribution and then destroys them once they leave the player's area. However, if the player pays attention to an NPC, the system creates an alibi that explains the current behavior and the character is controlled according to the alibi's distribution.

Similar to alibi generation, Initial State Revision (ISR) [26] allows narrative planners to dynamically reconfigure the initial state of a planning problem in order to better facilitate story creation. Initial state literals can be set to true or false, like in a normal PDDL problem, but can also be set to undetermined. An emergent narrative system, called the Virtual Storyteller [27], uses a concept called late commitment that is

similar to initial state revision. Late commitment is a feature where story characters can improvisationally change the state and rules of the story world simulation.

In order for our framework to take advantage of a player's incomplete knowledge of their environment, it needs a theory to predict what story characters observe and know about their world. The framework uses a microtheory [28] to create and maintain its model of player knowledge. A microtheory is a modular way to specify a model of some domain that can later be expanded or substituted for a more complete theory.

D. Procedural Content Generation

The final component of the framework is a procedural content generation (PCG) pipeline that creates a visual interface for the interactive narrative plan trees generated by mediation. Other systems have procedurally generated game world layouts that support a given narrative plan [29] and used planning models to procedurally generate game mechanics [30]. This system uses PCG to automatically create, configure, and maintain virtual worlds based on a PDDL representation to support story alibis in the game world.

III. Problem Description

This section defines a strong story interactive narrative problem that allows for multiple initial and goal states, multiple world mechanics in the form of operator libraries, and partial player knowledge of the story world.

A. World Representation and Dynamics

A strong story IN is an interactive story where non-player characters are controlled according to a narrative model. Interactive stories take place in a story world. In our model, story worlds consist of *locations*, *characters*, and *objects*, represented by PDDL *constants*. Characters are agents that act and effect change in the story world. A player is a special character controlled by a human participant. Character actions are represented by instantiated, fully ground PDDL *operators*. An operator is a tuple $\langle p, e, b \rangle$ of preconditions p, effects e, and bindings b that map variables to constants. We refer to some operator o's preconditions as p_o, its effects e_o, and its bindings b_o. Character *actions* are represented by fully ground operators. An action is called an *instantiation* of its abstract operator. If an action's preconditions hold in a given state, the action is *enabled*. Actions have deterministic effects.

All actions are performed by story characters. When action a is carried out by an agent g, we say g *performs* a and a's *actor* is g. Together, a story world state and an operator library describe a *state transition system* connected by enabled actions. In our system, players always have the ability to take enabled actions. Transition systems are represented with the tuple $\langle s, \omega \rangle$, where s is a state and ω is a set of operators. When an enabled action a is performed by an agent g in state s it produces a *successor state* s', where each ground atomic formula in s' matches s unless it is updated by an effect in e_a. We call a sequence of enabled actions and their corresponding successors in a transition system a *trajectory*. We refer to the sequence of actions in a trajectory t as α_t and the sequence of states in a trajectory σ_t. A trajectory *reaches* the final state in its sequence. Planning is a search for a trajectory that reaches a state where all goal conditions are enabled.

B. Interactive Narrative Planning

A *planning problem* describes a set of trajectories through a transition system to a set of goal states. Planning problems are comprised of a state transition system, which is an initial state and set of operators, and a set of goal conditions. Goal conditions are literals that must be true when the story ends. Problems are represented $\langle i, \gamma, \omega \rangle$ where i is the initial state description, γ is the goal conditions, and ω is the set of operators. A *plan* is a solution to a planning problem. It is a trajectory $t = (s_0, ..., s_n)$ through a state transition system $\langle i, \omega \rangle$ where $s_0 = i$ and $\gamma \in s_n$. A *solvable trajectory* is any trajectory from which a goal state can be reached by adding a further sequence of enabled actions. If a story world can be modeled as a transition system and interesting narrative properties can be specified by a set of goal conditions then story generation can be automated as planning. However, interactive narrative systems must also account for alternate courses of action that could be taken by a human participant.

In interactive narrative planning, one or more of the characters acting in the transition system is controlled by a human player. Given freedom to act, players may disrupt the system's plan by taking enabled actions that contradict the story trajectory. The goal of interactive narrative generation is to plan contingencies for deviations a player may perform. Our interactive narrative problem definition allows for multiple initial states, similar to Initial State Revision [31], multiple goal states, similar to ASD's fourth tier of replanning [21], and multiple operator sets, which can be used to implement intervention. Our *interactive narrative problem* is represented $\langle I, G, O \rangle$ where $I = \{i_0, i_1, ..., i_n\}$ is a set of initial states, $G = \{\gamma_0, \gamma_1, ..., \gamma_n\}$ is a set of goal states, and $O = \{\omega_0, \omega_1, ..., \omega_n\}$ is a set of operator sets.

For each interactive narrative problem a set of *actions*, *trajectories*, *solvable trajectories*, *states*, and *plans* can be derived. These sets bound what events, states, and solutions can occur during interaction. An interactive narrative's *action set*, A, consists of every valid binding of each operator $o \in \omega \in O$ using every combination of constants in each $i \in I$. An interactive narrative's *trajectory set*, T, consists of all possible trajectories from each $i \in I$ using each $\omega \in O$. The set of trajectories describes every possible way the state transition systems bound by the problem can evolve over time. An interactive narrative's *solvable trajectory set*, V, consists of all solvable trajectories from each $i \in I$ using each $\omega \in O$ to every possible goal state containing a $\gamma \in G$. The set of solvable trajectories describes every possible sequence of actions from which a goal state can be reached. An interactive narrative's *state set*, S, consists of all possible states reached by some trajectory $t \in T$. The set of states describes every possible encounterable state. Finally, An interactive narrative's *plan set*, P, consists of all trajectories that reach a goal state.

Interactive narratives do not exist in isolation, they are meant to be performed with a player in an interactive envi-

ronment. An interactive narrative environment can be defined as an interface that exposes the state transition system defined by an interactive narrative problem to a player.

C. Interactive Narrative Play

One way to conceptualize the possible ways a state transition system can evolve over the course of an interactive narrative is to enumerate its trajectories as a tree. An interactive narrative's *world tree*, w, is a tree whose vertices are states in S and edges are actions in A that correspond to a single set of operators $\omega \in O$. The root of the tree is some initial state $i \in I$. The outgoing edges of any vertex s is the set of actions $\alpha \in A$ that correspond to ω and are enabled in s. Edges in w correspond to some action α and lead from a predecessor state s to a successor state $s' = e_a \cup s$. A path from the root of the tree to any vertex is equivalent to a trajectory $t \in T$. A fully expanded world tree represents every path through the state transition system $\langle i, \omega \rangle, i \in I, \omega \in O$.

There is a world tree for every combination of initial states and operator libraries in the interactive narrative problem. An interactive narrative's collection of world trees, called a *world forest*, W, is a graph containing all its possible world trees, $W = \langle I, O \rangle = \{\langle i_0, \omega_0 \rangle, \langle i_0, \omega_1 \rangle, ..., \langle i_n, \omega_n \rangle\}$. An interactive narrative's world forest will contain a unique tree for every transition system created by an initial state-operator library pairing $\langle i, \omega \rangle$ specified by the IN problem. *Gameplay* is the iterative navigation of a world tree w in the world forest of an interactive narrative problem that begins at the root of w and moves downward along edges to successor states as actors take enabled actions according to a pre-established turn ordering. During gameplay, the *world history* is the trajectory taken through the world tree. Gameplay's *current state* is the one reached by the world history trajectory.

Interactive narratives require participation from a player. The system must offer an interface to expose state configurations and allow the player to take enabled actions in order to provide gameplay. A *game world* is an environment that provides an interface for players to act as agents and perform gameplay on a world tree w. The game world exposes a current state s and allows players to take enabled actions that correspond to outgoing edges of s in w. Clients may not have complete knowledge of the story world during gameplay. This incomplete knowledge can be used to further the system's goals. The final component of the system definition is a model of what characters observe as they interact in the game.

D. Agent Knowledge

Most story worlds are only partially observable to their characters, so we track character knowledge with a microtheory. A microtheory is an extensible model of some domain that can be changed or replaced without affecting the overall system. Building a robust model of how characters come to know the world around them is outside the scope of this system, so we use a microtheory to predict character knowledge.

The current microtheory is a set of rules that asserts characters observe all actions and the state of all objects they are co-located with. This microtheory does not model more intricate details of psychology like the distinction between knowledge and belief or between an event or object's visibility and whether it is actually observed. However, a microtheory's rules can be extended or replaced without affecting the overall system's use of the model. The current model is overly-protective of possible player observations, but a more robust model can be easily substituted in at a later time to more accurately represent what a player knows and give the system more control.

A *knowledge microtheory* is a collection of axioms Mt that, given a history h and an agent g, describe a subset of actions in α_h and a subset of atomic formula in each $s \in \sigma_h$ that g has observed. The particular axioms of Mt defines a trajectory for each agent participating in the interactive narrative. During gameplay, an agent g's *knowledge* is a trajectory k_g that describes what g knows about the world history h given a knowledge microtheory Mt. k_g is a subset of h where any atomic formula or action determined by Mt to be unobserved by g is substituted with $\emptyset$. An agent's knowledge of the world history describes a *superposition* of possible trajectories that may have occurred. During gameplay, an agent g's *world history superposition* is a set, H_g, of trajectories that could possibly be the world history from an agent g's perspective according to Mt. A trajectory $t \in T$ belongs to H_g if for every $s_i \in \sigma_t$ and $s'_i \in \sigma_{k_g}, s'_i \subset s_i$ and $a'_i \in \alpha_{k_g}, a_i = a'_i$ or $a'_i = \emptyset$. There could be trajectories in H_g that correspond to paths through multiple trees in W if the agent has limited knowledge of h's initial state or operator library.

Any trajectory that belongs to an agent's superposition is consistent with what the character knows according to the microtheory. A trajectory t is *consistent* with an agent g's knowledge if $t \in H_g$. The trajectories in H_g describe a set of possible world states the agent g could exist in. During gameplay, an agent g's *world state superposition* is a set $S_g = \{s_0, s_1, ..., s_n\}$ of states that could possibly be the current world state from an agent g's perspective according to Mt. A state $s \in S$ belongs to S_g if it is reached by a trajectory $t \in H_g$. Given the world history superposition of every player participating in the interactive narrative, the system's task of maintaining a solvable trajectory can be defined.

E. Perceptual Experience Management

A trajectory is both solvable and consistent if a goal state can be reached from the trajectory and the trajectory belongs in the world history superposition of an agent. During gameplay, a consistent and solvable trajectory for agent g is any trajectory t where $t \in V, H_g$. This set of consistent and solvable trajectories for an agent g during gameplay is referred to as VH_g. The task of a perceptual experience manager is to ensure that each player has at least one solvable trajectory consistent with their observations. In other words, for every player agent g ensure $VH_g \neq \emptyset$. The next section presents ensuring that $VH_g \neq \emptyset$ holds in more situations by allowing interactive narrative systems to compute all trajectories in H_g.

IV. Mediation Framework

This section describes a framework that builds world trees from an interactive narrative problem for a single player,

exposes world tree data structures through a PCG interface, and transitions players between world histories to reach goals.

The last section formally defines the task of automated, strong story interactive narrative systems as ensuring at all times during gameplay, for all players g, $VH_g \neq \emptyset$. The section defines this task in terms of abstract sets. In practice, these sets are large and cannot be fully generated. The first hurdle to overcome when working towards solving the interactive narrative task is to find trajectories that belong in VH_g without expanding all possibilities. This section presents a plan-based approach to finding members of VH_g through search. This framework makes two improvements over existing plan-based experience managers that allow it to ensure $VH_g \neq \emptyset$ in more gameplay situations than previous approaches by searching through alternate possible world histories and mechanics that are consistent with player knowledge.

Previous plan-based interactive narrative systems do not utilize a model of player knowledge when building branching story data structures. This artificially restricts H_g to have a single trajectory, which is the exact series of events that have played out in the story world. This restriction holds if the player has full knowledge of story events. However, when the player has incomplete knowledge of the story world, this model limits VH_g to trajectories that begin with a single member of H_g. So widening H_g by considering all consistent trajectories according to a model of player knowledge strictly increases the trajectories in VH_g and ensures that $VH_g \neq \emptyset$ in more situations. Section IV-B presents a plan-based algorithm called *event revision* to search all possible trajectories in VH_g, given a microtheory, player character, and a world history.

Previous plan-based systems also do not utilize a model of player knowledge when executing interventions. One type of intervention is a shift between two alternate models of world mechanics to prevent an unwanted outcome of a player's action. Interventions in plan-based systems can be represented as shifts between alternate sets of operators that represent similar actions but have different sets of preconditions or effects. If the experience manager is allowed to jump back and forth between operators at will, it is commonly thought that the player will notice inconsistencies in the way the world operates. Due to this, systems like ASD [21] remove interventions entirely. However, if the experience manager only shifts between operator libraries when both are consistent with what the player has experienced, then the player should not notice an inconsistency. Intervening to cancel out effects of player actions that contradict author goals is a second way to ensure that $VH_g \neq \emptyset$ in more situations. Section IV-C presents a plan-based algorithm called *domain revision* that identifies and restricts interventions inconsistent with player knowledge.

A. Mediation

Our interactive narrative system is a plan-based *mediator*. Mediators are strong-story [1] interactive narrative agents that build branching story structures by generating a linear narrative and then creating contingencies for each way a player-controlled character could disrupt the planned story. The input to mediation is a narrative planning problem created by an author. Given a planning problem and a planner, our mediator generates a *mediation game tree*, a world tree where there is exactly one player-controlled character, each character is given a sequential turn order for taking action, and a plan is associated with every vertex. The root of the tree is the initial state. The outgoing edges of any vertex is the set of actions enabled in the vertex's state performed by the character with the current turn. Actions that non-player characters take are dictated by the plan associated with the outgoing vertex. The plan associated with the root node is the original solution to the input planning problem given by the planner.

As new levels of the mediation game tree are expanded, the original plan is updated at each new node. Instead of generating a new plan at every expanded vertex, our mediator utilizes a plan management strategy. To manage plans, the system first draws *dependencies* between actions in the current plan. A dependency is an interval in a trajectory t that begins at a state s, ends at an action a_j, and is associated with an atomic formula p where p is a precondition of a_i and s is the successor of the last action a_i that is ordered before a_j in t and has p as an effect. If no such action exists, s becomes the initial state s_0. For every precondition of every action in the current plan, the system draws a corresponding dependency. From the plan and set of dependencies, the system can classify the vertex's outgoing actions as *constituent*, *consistent*, or *exceptional*.

A *constituent* action is one that is prescribed by the current plan. To update the plan after a constituent action, the system removes the taken action from the trajectory and updates the dependencies. A *consistent* action is one whose effects do not reverse any atomic formula attached to a dependency in the initial state. When a consistent action is performed, the plan does not change. An *exceptional* action is one with at least one effect that reverses an atomic formula attached to a dependency in the initial state. When an exceptional action is taken, the current plan must be revised by the planner. If the planner does not return a solution, the game tree has entered the set of non-valid trajectories from which there is no return.

As described, mediation is over-constrained to consider only plans that match the actual path through the mediation game tree, when the player may exist in a superposition of possible consistent trajectories given their partial knowledge of the game world. Two methods allow the system to transition the player between all trajectories in the player's set of possible world histories H_g: *event revision* and *domain revision*. The first method, event revision, searches through alternate world histories compatible with what the player has observed.

B. Event Revision

Event revision [32] finds trajectories with alternate histories in H_g instead focusing on a single default path. Event revision is most useful when many characters have the opportunity to take meaningful actions while the player is not observing. This event revision process is planner independent. It is presented modularly and can be integrated into a planner's action selection process. There are two phases of event revision: *removal*, where a subjective history is created by removing actions and state formulae unobserved by the player, and *planning*, where

the system searches for a new plan that is compatible with the remaining player observations.

Algorithm 1 Unobserved Element Removal

```
REMOVE (World History Trajectory h,
        Knowledge Microtheory Mt,
        Story Agent g)
  trajectory t ← h
  for each action a ∈ α_t
    if g does not observe a according to Mt
      a ← ∅
  for each state s ∈ σ_t
    for each atomic formula f ∈ s
      if g does not observe f according to Mt
        s ← s − f
  return t
```

The removal phase takes place before the system searches for a plan. An agent knowledge trajectory, k_g, is created where all unobserved actions and atomic state formulae in the world history are removed. Pseudocode for the action removal phase is given in Algorithm 1. The trajectory returned by Algorithm 1 is the player g's knowledge of world history events, k_g.

Algorithm 2 Enabled Action Constraint

```
CONSTRAIN (Current Trajectory t,
           Agent Knowledge k_g,
           Knowledge Microtheory Mt,
           Story Agent g,
           Current Actor According to Turn Ordering c_a)
  s ← state reached by t
  α ← set of actions enabled in s
  if |α_t| > |α_{k_g}|  return α
  for each action a ∈ α
    s_a ← successor state created by applying a to s
    t_a ← trajectory created by appending (a, s_a) to t
    if Consistent(Remove(t, Mt, g), k_g) is false
     or a's actor is not c_a
      α ← α − a
  return α
```

The second phase uses a modified planning algorithm to search for a solution plan from the initial state of the problem at the root of the tree. The planning algorithm is modified to constrain enabled actions prior to the current state of gameplay to those that are consistent with the player's observations and the tree's ordering. Pseudocode for constraining actions is given in Algorithm 2 and a sub-process that determines consistent trajectories is given in Algorithm 3.

These algorithms enable the system to explore all possible world histories consistent with player observations. It can also be used to move between trees in the problem's world forest to paths that start with alternate initial states. However, the system is still over constrained since it can only use the single domain model the framework begins with. The second component, domain revision, transitions between different domains consistent with player observations.

Algorithm 3 Perceptually Consistent Trajectory Check

```
CONSISTENT (Current Trajectory t, Agent Knowledge k_g)
  for each action a_t ∈ α_t at position i
    if action a_{k_g} ∈ α_{k_g} at position i is not ∅
     and not equal to a_t  return false
  for each state s_t ∈ σ_t at position i
    if s_t is not equal to state s_{k_g} ∈ σ_{k_g}
     at position i  return false
  return true
```

C. Domain Revision

Domain revision transitions between alternate trees in the world forest W that correspond to world histories with alternate, compatible operators given in O. Two sets of operators are compatible, given player observations, if both support the same series of observed actions and precondition-effect pairs of observed actions. Domain revision is most useful when the player is learning how the game world works and there are many possible world models the system can choose between. If a player is replaying a game for the second time, domain revision should retain its model of what game mechanics the player has observed if it wants to maintain the illusion of a fixed story world. When combined with event revision, domain revision allows the full space of VH_g to be searched.

Domain revision is performed with the same three components that enable event revision. Additional trees are explored by moving to other domains in O if the current one fails. When a domain is selected, planning fails if the domain is incompatible with player observations. The process exhaustively searches domains in O until a plan is found or O is exhausted. A problem with this approach is it doesn't reason about what domains should be considered which may make online domain revision impractical. A solution is to constrain the search to domains most likely to mediate the exceptional action.

One way to constrain the space is to only alternate domains that change the behavior of the exceptional action template [33]. A single operator template should have multiple alternate sets of preconditions and effects for different domains in O. The system can perform domain revision using operator libraries that differ in their definition of the exceptional action's operator template. This method corresponds to a class of interventions where the outcome of an action is changed to prevent it from being exceptional.

V. The General Mediation Engine

This section presents details of a plan-based architecture for interactive narrative generation called the General Mediation Engine (GME) that implements the mediation components detailed in Section IV. GME creates gameplay by maintaining data structures that correspond to sets outlined in Section III, providing a user interface to the data structures through procedural content generation, and utilizing planning to search for trajectories in the user's VH set. Narrative can be viewed as consisting of story and discourse [34]. Similarly, GME can be viewed as having a *story generator*, which produces interactive narrative data structures, and a *discourse generator*,

which generates a representation of the story world. GME is implemented in C# [35], [36]. Its story generator takes as input an interactive narrative problem and builds a mediation game tree. It maintains a world state and a plan and updates its data structures according to actions taken by the player and NPCs. The story generator exposes information to the discourse generator which conveys information to the player.

GME assembles interfaces with a procedural content generation (PCG) pipeline to display interactive narratives to players. GME has three text interfaces: a local and web-based console application and a hyptertext Choose Your Own Adventure. It also has a 2D interface built with the Unity game engine. Each of these uses a library of assets to dynamically configure an interface based on GME's current state. These configurations are done automatically given a PDDL domain and problem file so a new gameplay experience can be generated from each new PDDL input.

```
Welcome to ZOMBIE

You are standing in the livingroom. You see Linda. You see cabinet. cabinet is a
 closed and is locked. You see doors leading to the Bedroom, Outside, and the Ce
llar.

> move bedroom

.....

You move bedroom
     Ash is not at the Livingroom
     Ash is at the Bedroom

You are standing in the bedroom. You see key. The key is old. You see a door lea
ding to the Livingroom.

>
```

Fig. 1. GME's online console interface.

GME's first interface is a windows console application [37]. The interface uses simple text templates to convey world states and character actions from the GME back end to the player. The player advances gameplay by typing actions into the text input. When the player types in an enabled action, the interface informs GME, which transitions down the action's edge in its tree. GME then takes action for its system-controlled characters according to the current plan. Once finished, the interface displays observed system actions along with the new world state to the player. GME also has two online versions of this interface: one shown in Figure 1 and a Choose Your Own Adventure (CYOA) web page interface.

The final interface is a visual 2D system implemented in the Unity game engine [38], called UGME. Figure 2 pictures a demo sneaking game implemented in UGME called Base Case [39]. Instead of conveying world states and character actions with text templates, UGME constructs and controls a 2D environment from a GME DLL using the Unity game engine. Before execution, collections of assets including art, animations, sound, and code, called prefabs can be compiled in Unity by an author. UGME instantiates prefabs from its resource library to represent story objects in GME's story world. The entire *Base Case* game world is created with prefabs instantiated from this resource library.

VI. Evaluation

This section presents an evaluation of event revision and domain revision as implemented in GME by comparing metrics recorded from expansions of interactive narrative trees. The first tree in each test is generated with GME's baseline mediation method. The baseline tree's metrics are then compared against a second tree that is generated with either event revision, domain revision, or both working together. Event and domain revision monotonically add nodes to the mediation game tree per level, so we can expect the number of nodes these trees to always be equal or more than the number of nodes per level in the baseline. Sections VI-A, VI-B, and VI-C all provide evidence that supports this hypothesis by examining trees built from three different PDDL inputs.

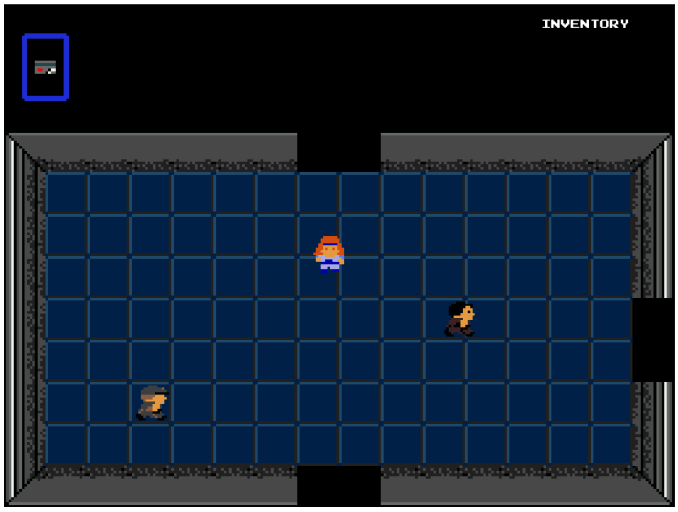
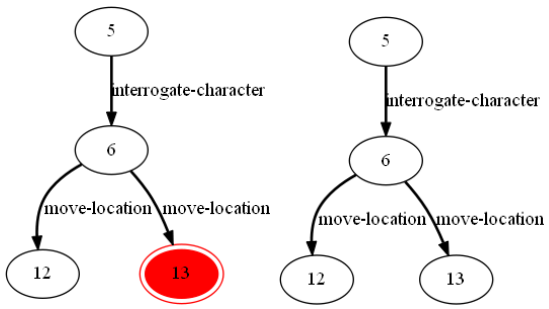


Fig. 2. UGME gameplay.

A. Batman Domain

The first test domain is a situation set in a Batman universe. The problem first appeared as an example for event revision in the context of POCL mediation [40], [32]. The Batman domain models a situation in the film *The Dark Knight*. The problem is set in Gotham City. The Joker is being held at the Gotham Police Department. The player is Batman and is about to confront the Joker. The Joker's henchmen have kidnapped two important characters, Rachel Dawes and Harvey Dent, and are transporting them to hostage locations. The Joker tells Batman about his hostages. The Joker says the hostages will be killed and Batman has time to only save one. In order to set up the final act where Harvey Dent is turned into Two Face, the system has the goal of Batman saving Harvey at the expense of Rachel. For this to happen, the system has the henchmen deliver Rachel to 52nd Street and Harvey to Avenue X. The Joker tells Batman where the characters are and Batman decides who to save. In the film, Batman travels to Avenue X where he saves Harvey.

However, these world mechanics allow Batman to travel to 52nd Street and save Rachel, which would kill Harvey and end the possibility of a final act with Two Face. If event revision is used, the past events of the henchmen transporting Rachel and Harvey to their destinations are identified as outside the players knowledge of the world history. Once these events are identified they can be removed and replaced with the alternate



(a) Baseline Tree (b) Event Revision Tree

Fig. 3. Graph of possible player actions in a baseline and event revision tree created by GME from the Batman domain. The trees show only player actions, not NPC behaviors. States are nodes. Possible player actions are directed edges. A dead end is indicated with an outlined, red circle.

event sequence of the henchmen delivering Rachel to Avenue X and Harvey to 52nd Street. In this history, Joker lies about hostage locations in order to reverse Batman's decision.

We used GME to generate a baseline and an event revision tree for the Batman Domain. Our hypothesis is that GME will generate equal or more nodes per level in its event revision tree when compared to the baseline. The first 18 levels of each tree were generated, the point where a goal node is first encountered. As expected, event revision produces 23 valid nodes as opposed to the baseline method's 17 nodes. Level 12 is where the player decides whether to move to Avenue X or 52nd Street. From here until the story ends at Level 18 the event revision tree has 100% more branches (2 to 1) than the baseline. Figure 3 shows the two trees. A single dead end is reached in the baseline, where Rachel is saved, but no dead ends are reached in the event revision tree.

B. Wild West Domain

The second domain is set in a generic Wild West world. The player is a gun for hire who has been captured by a band of thieves. The thieves are holding the player prisoner in a small house outside the gang's main compound. The player's hands are tied by rope and a single bandit stands guard outside the house by a bonfire. Both the player and the guard have a concealed knife. The system's goal is to solidify trust between the player and a potential love interest who works with the thieves. The player met the love interest earlier in the game and was surprised to find the love interest at the camp before the player was captured. To solidify this trust, the system plans for the love interest to apprehend the guard, tie the guard's hands with rope, and release the player. The system's goal is for the player and love interest to escape together and return to the nearest town. To force this situation, the guard will cut his bonds, return to the compound, and warn his fellow bandits.

However, these world mechanics allow the player to cut their own bonds and escape without the love interest. Using intervention, the system could prevent the player from cutting through the rope by replacing the original effect with one that does not update the world state. It's not unreasonable to show the player being unable to reach the concealed knife or cut the rope. But if the player emerges from the house with their love interest to find ropes cut by the guard in order to escape, they might realize the system has created a double standard in which ropes can be cut by bound persons, but only when the system allows it. If domain revision is used, the rule - bound persons with a concealed knife can cut through their bonds - is erased from the domain when the intervention is used against the player. From that point on, no character would be able to free themselves with a concealed knife when bound with rope. Instead, the guard might burn his bonds using the bonfire and leave behind charred rope for the player to find.

We used GME to generate a baseline and a domain revision tree for the Wild West Domain. Our hypothesis is that GME will generate equal or more nodes per level in its domain revision tree when compared to the baseline. The first 11 levels of each tree were generated, the point where a goal node is first encountered. As expected, domain revision produces 36 valid nodes as opposed to the baseline method's 10 nodes. Levels 3, 6, and 9 are where the player makes a decision whether to sit still or make an escape. After Level 3 the domain revision tree has 100% more branches (2 to 1), after Level 6 the domain revision tree has 400% more branches (4 to 1), and after Level 9 the domain revision tree has 800% more branches (8 to 1) when compared to the baseline tree. Figure 4 shows a single dead end is reached in the accommodation tree at Level 3, 6, and 9, where the player has an opportunity to escape, but no dead ends are reached in the domain revision tree.

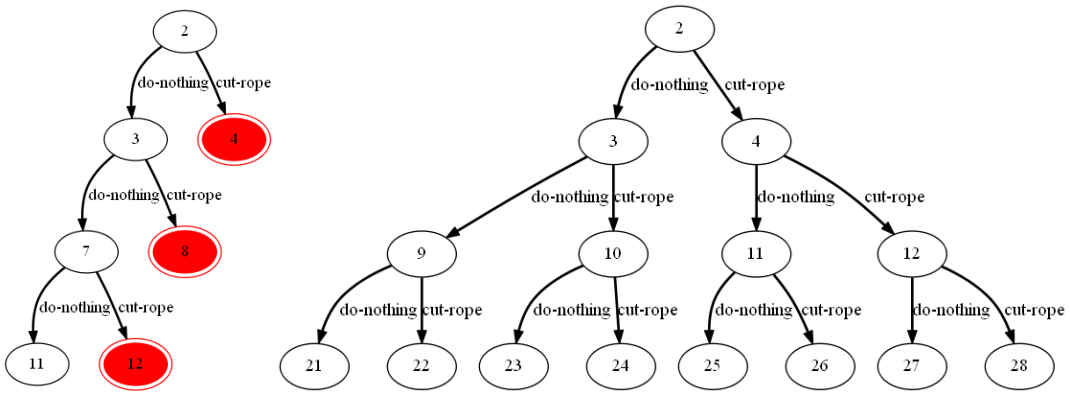
C. Spy Domain

The final domain is set in a Spy world loosely based on the *Metal Gear* game series. The player, a spy named Snake, must foil the final attempt of the antagonist, the Boss, to bring a weaponized satellite online. The confrontation takes place on a satellite dish antenna cradle with five discrete locations where the Snake and Boss interact: the Elevator Room, Gear Room, Left and Right Walkways, and the Platform. The locations are connected by doors that can only be traversed in one direction. The Elevator Room leads to the the Gear Room, the Gear Room leads to the Left and Right Walkway, and each Walkway leads to the Platform. Snake begins the game in the Elevator Room. Her job is to disable the antenna's alignment mechanism in the Gear Room and eliminate the Boss. The Boss plans to send instructions to the satellite by linking his phone to one of four computer terminals on the cradle. The author's goal is for the Boss to set a trap to be disabled by the player before a final confrontation on the Platform.

TABLE I
SPY DOMAIN PERCENTAGE INCREASE IN BRANCHES OVER BASELINE

Depth	Domain	Event	Domain + Event
4	5	5	10
5	10	12	22
6	16	23	39
7	22	27	48
8	28	29	57

We used GME to generate a baseline, event revision, domain revision, and combined event and domain revision interactive narrative tree for the Spy Domain. Our hypothesis is that GME



(a) Baseline Tree (b) Domain Revision Tree

Fig. 4. Graph of player choices in a baseline and a domain revision tree automatically created by GME from the Wild West domain.

will generate equal or more nodes per level in its event and domain revision trees when compared to the baseline. The first 8 choice levels were generated, past the point where a goal node is first encountered. As expected, domain revision produces 5290 valid nodes, event revision produces 5415 valid nodes, and both produce 6431 valid nodes as opposed to the baseline method's 4276 nodes. Table I shows the percentage increase in valid branches from choice levels 4 to 8 in the domain, event, and combined trees when compared to the baseline. The increase in branches grows over time in each tree and reaches over 50% in the combined tree at Level 8.

TABLE II
SPY DOMAIN AVERAGE TIME IN SECONDS PER NODE EXPANDED

Baseline	Domain	Event	Domain + Event
0.75	1.23	1.42	1.90

Table II shows the average time in seconds it takes GME to generate a node in each tree. In all cases the time is below 2 seconds per node, which can be lessened further by caching. This is fast enough to generate and cache single successor nodes online as the player interacts with GME.

TABLE III
SPY DOMAIN AVERAGE SUCCESSFUL CHOICE OPTIONS PER CHOICE

Baseline	Domain	Event	Domain + Event
3.05	3.17	3.17	3.27

Table III shows the average number of choice options per player choice that match author goals in each of the four trees. The number, averaged over thousands of possible choice points, grows with each perceptual experience management method used. Similarly, Table IV shows the average number of choice options per player choice that lead to dead ends in each of the four trees. The number declines with each perceptual experience management method used.

VII. Future Work

This section explores three areas of work to be expanded on: a human subjects study, the model of player knowledge, and GME's PCG pipeline. First, we can test whether human participants notice manipulations perceptual experience management performs with event and domain revision. We plan a human subjects study where participants play GME games to determine whether they notice manipulations. A control game can make modifications that contradict the model of player knowledge. We predict players will notice control game manipulations but not in the regular GME games.

TABLE IV
SPY DOMAIN AVERAGE UNSUCCESSFUL CHOICE OPTIONS PER CHOICE

Baseline	Domain	Event	Domain + Event
1.39	1.29	1.17	1.10

Next, the model of user knowledge can be expanded. Our microtheory predicts users perfectly observe everything they are co-located with. However, players may not see or remember everything that is presented on screen. Refinement of the model may expand the number of manipulations perceptual experience management can perform. Finally, GME could use a general visual pipeline for translating PDDL states and mechanics into a playable interface. The current system uses prefabs, a predefined generator, and action interfaces that convey player actions to GME. A full PCG pipeline would automatically determine these components by generating art assets that represent PDDL and assemble a game world and mechanics from the declarative input.

VIII. Conclusion

This document presents perceptual experience management, a framework for widening the branching factor of interactive narrative trees produced by automated, strong story experience management systems. This is achieved by tracking player observations during gameplay and dynamically revising past story events and world mechanics to allow player actions while maintaining author control and maintaining the illusion of consistency from the player's perspective. We show perceptual experience management generates interactive narrative trees with more branches that allow player choices and maintain author constraints when compared to a baseline.

REFERENCES

[1] M. Riedl and V. Bulitko, "Interactive Narrative: An Intelligent Systems Approach," *AI Magazine*, vol. 34, no. 1, pp. 67–77, 2013.

[2] B. Magerko, "Evaluating Preemptive Story Direction in the Interactive Drama Architecture," *Journal of Game Development*, vol. 2, no. 3, pp. 25–52, 2007.

[3] M. O. Riedl and R. M. Young, "From Linear Story Generation to Branching Story Graphs," *Computer Graphics and Applications*, vol. 26, no. 3, pp. 23–31, 2006.

[4] ——, "Narrative Planning: Balancing Plot and Character," *Journal of Artificial Intelligence Research*, vol. 39, no. 1, pp. 217–268, 2010.

[5] S. G. Ware and R. M. Young, "Glaive: A State-Space Narrative Planner Supporting Intentionality and Conflict," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2014, pp. 80–86.

[6] M. Riedl, C. J. Saretto, and R. M. Young, "Managing Interaction Between Users and Agents in a Multi-Agent Storytelling Environment," in *International Conference on Autonomous Agents and Multiagent Systems*, 2003, pp. 741–748.

[7] M. Bal, *Narratology: Introduction to the Theory of Narrative*. University of Toronto Press, 2009.

[8] R. M. Young, S. G. Ware, B. Cassell, and J. Robertson, "Plans and Planning in Narrative Generation: A Review of Plan-Based Approaches to the Generation of Story, Discourse and Interactivity in Narratives," *Sprache und Datenverarbeitung*, vol. 37, no. 1-2, pp. 41–64, 2013.

[9] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, *PDDL - The Planning Domain Definition Language*, 1998.

[10] P. Haslum, "Narrative Planning: Compilations to Classical Planning," *Journal of Artificial Intelligence Research*, vol. 44, pp. 383–395, 2012.

[11] J. Teutenberg and J. Porteous, "Efficient Intent-Based Narrative Generation Using Multiple Planning Agents," in *International Conference on Autonomous Agents and Multi-Agent Systems*, 2013, pp. 603–610.

[12] S. G. Ware and R. M. Young, "CPOCL: A Narrative Planner Supporting Conflict," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2011, pp. 97–102.

[13] J. Hoffmann, "FF: The Fast-Forward Planning System," *AI Magazine*, vol. 22, no. 3, p. 57, 2001.

[14] M. Helmert, "The Fast Downward Planning System," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[15] E. Packard, *The Cave of Time*, ser. Choose Your Own Adventure. Bantam Books, 1979.

[16] R. Aylett, "Narrative in Virtual Environments - Towards Emergent Narrative," in *AAAI Fall Symposium: Narrative Intelligence*, 1999, pp. 83–86.

[17] R. S. Aylett, S. Louchart, J. Dias, A. Paiva, and M. Vala, "FearNot! – an Experiment in Emergent Narrative," in *International Workshop on Intelligent Virtual Agents*, 2005, pp. 305–316.

[18] M. Mateas and A. Stern, "Façade: An Experiment in Building a Fully-Realized Interactive Drama," in *Game Developer's Conference*, vol. 2, 2003.

[19] J. Bates, "The Nature of Characters in Interactive Worlds and the Oz Project," Carnegie Mellon University, Tech. Rep. CMU-CS-92-200, 1992.

[20] 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*, vol. 1, no. 1, pp. 51–70, 2004.

[21] M. O. Riedl, A. Stern, D. M. Dini, and J. M. Alderman, "Dynamic Experience Management in Virtual Worlds for Entertainment, Education, and Training," *International Transactions on Systems Science and Applications*, vol. 4, no. 2, pp. 23–42, 2008.

[22] M. O. Riedl, "Towards Integrating AI Story Controllers and Game Engines: Reconciling World State Representations," in *IJCAI Workshop on Reasoning, Representation and Learning in Computer Games*, 2005.

[23] J. Porteous, M. Cavazza, and F. Charles, "Applying Planning to Interactive Storytelling: Narrative Control Using State Constraints," *Transactions on Intelligent Systems and Technology*, vol. 1, no. 2, p. 10, 2010.

[24] A. Ramirez and V. Bulitko, "Automated Planning and Player Modeling for Interactive Storytelling," *Transactions on Computational Intelligence and AI in Games*, vol. 7, pp. 375–386, 2015.

[25] B. Sunshine-Hill and N. I. Badler, "Perceptually Realistic Behavior through Alibi Generation," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2010, pp. 83–88.

[26] M. O. Riedl and R. M. Young, "Open-World Planning for Story Generation," in *International Joint Conference on Artificial Intelligence*, 2005, pp. 1719–1720.

[27] I. Swartjes, E. Kruizinga, and M. Theune, "Lets Pretend I Had a Sword," pp. 264–267, 2008.

[28] R. Guha and J. McCarthy, "Varieties of Contexts," in *International and Interdisciplinary Conference on Modeling and Using Context*, 2003, pp. 164–177.

[29] K. Hartsook, A. Zook, S. Das, and M. O. Riedl, "Toward Supporting Stories with Procedurally Generated Game Worlds," in *Conference on Computational Intelligence and Games*. IEEE, 2011, pp. 297–304.

[30] A. Zook and M. O. Riedl, "Automatic Game Design via Mechanic Generation," in *AAAI Conference on Artificial Intelligence*, 2014, pp. 530–537.

[31] S. G. Ware and R. M. Young, "Rethinking Traditional Planning Assumptions to Facilitate Narrative Generation," in *AAAI Fall Symposium: Computational Models of Narrative*, 2010, pp. 71–72.

[32] J. Robertson and R. M. Young, "Finding Schrödinger's Gun," in *Artificial Intelligence and Interactive Digital Entertainment*, 2014, pp. 153–159.

[33] ——, "Interactive Narrative Intervention Alibis through Domain Revision," in *Workshop on Intelligent Narrative Technologies*, 2015, pp. 49–52.

[34] S. B. Chatman, *Story and Discourse: Narrative Structure in Fiction and Film*. Cornell University Press, 1980.

[35] J. Robertson and R. M. Young, "Gameplay as On-Line Mediation Search," in *Experimental AI in Games Workshop*, 2014, pp. 42–48.

[36] "The General Mediation Engine," https://github.com/justusrobertson/GME, accessed: 2017-08-21.

[37] J. Robertson and R. M. Young, "The General Mediation Engine," in *Experimental AI in Games Workshop*, 2014, pp. 65–66.

[38] ——, "Automated Gameplay Generation from Declarative World Representations," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2015, pp. 72–78.

[39] M. Cook, S. Eiserloh, J. Robertson, R. M. Young, T. Thompson, D. Churchill, M. Cerny, S. P. Hernandez, and V. Bulitko, "Playable Experiences at AIIDE 2015," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2015, pp. 227–231.

[40] J. Robertson and R. M. Young, "Modeling Character Knowledge in Plan-Based Interactive Narrative to Extend Accomodative Mediation," in *Workshop on Intelligent Narrative Technologies*, 2013, pp. 93–96.

Justus Robertson holds a B.S. in Computer Science (2010) and English (2010), an M.S. in Computer Science (2014), and a Ph.D. in Computer Science (2017) from North Carolina State University, Raleigh, NC. He is currently a Postdoctoral Research Scholar in the Department of Computer Science at NCSU. His work is at the intersection of artificial intelligence, interactive narrative generation, procedural content generation, and automated game design, grounded in models of human cognition.

R. Michael Young (M '98; SM '07) received the B.S. degree in computer science from the California State University at Sacramento, Sacramento, in 1984, the M.S. degree in computer science with a concentration in symbolic systems from Stanford University, Stanford, CA, in 1988, and the Ph.D. degree in intelligent systems from the University of Pittsburgh, Pittsburgh, PA, in 1998. He worked as a Postdoctoral Fellow at the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, and as a Professor of Computer Science and Co-Director of the Digital Games Research Center, North Carolina State University, Raleigh, NC. He is currently a Professor in the School of Computing and the Deputy Director of the Entertainment Arts and Engineering Program, at the University of Utah, Salt Lake City, UT.