A Model of Superposed States

Justus Robertson
Department of Computer Science
North Carolina State University
Raleigh, NC 27695
jjrobert@ncsu.edu

R. Michael Young
School of Computing
The University of Utah
Salt Lake City, UT 84112
young@cs.utah.edu

Abstract

Interactive narratives (IN) are stories that branch and change based on the actions of a participant. A class of automated systems generate INs where all story branches conform to a set of constraints predefined by an author. Participants in these systems may create invalid branches by navigating the story world outside the bounds of an author's constraints. We approach this problem from an adversarial game perspective, where the IN system's goal is to prevent the player from creating invalid branches. From this perspective, one way an IN system can take action is to transition the game world between alternate states that are consistent with the player's observations during gameplay. In this paper we present a method of modelling and updating sets of world states consistent with player knowledge as a single superposed data structure. We discuss how this data structure can be used in an IN framework to maximize the probability that author constraints are maintained during gameplay.

Introduction

Interactive narratives are participatory stories whose events change based on actions a player takes during gameplay. A popular example of interactive narratives are the *Choose Your Own Adventure* (Packard 1979) series of game books. One open problem in interactive narrative design is the combinatorial explosion of possible stories based on the number of unique choices a player is able to make (Bruckman 1990). Interactive narrative agents, called *experience managers*, can mitigate this combinatorial explosion by automating the generation and control of character behaviors and plot structures. One type of experience management system is called a *strong story agent* (Riedl and Bulitko 2013). Strong story agents control story characters by building plots with interesting narrative properties. Interesting narrative properties can be specified as a set of constraints, called *authorial constraints*, on the possible plots generated by the system (Riedl, Thue, and Bulitko 2011).

One open problem in strong story systems is called the *boundary problem* (Magerko 2007) or *narrative paradox* (Louchart and Aylett 2003). This problem arises when the player takes a sequence of actions that navigates the story world into a state such that no plot exists that satisfies authorial constraints. There are several ways for a strong story system to deal with this problem, including action intervention, constraint exchange, and Schrödinger accommodation.

Intervention is a method of exchanging the effects of a player action that navigates the story world outside the bounds of authorial constraints with a second set that do not contradict the plot (Riedl, Saretto, and Young 2003). One potential problem with intervention is that repeated uses may decrease the system's *invisibility* (Roberts and Isbell 2008) by making the user aware there is a system manipulating the story world. A second method allows domain authors to specify multiple sets of constraints on the stories told by the system. The experience manager can continue to generate plots as long as they match at least one set of constraints (Riedl et al. 2008). One drawback of this method is it requires a larger authorial burden on the system designer to specify multiple sets of interesting constraints.

The final method, Schrödinger accommodation, finds new plots that match authorial constraints by searching through a set of possible histories that are consistent with the player's experience (Robertson and Young 2014a). One drawback of this approach is that it waits until the user tries to navigate the world outside the bounds of authorial constraints to act. In this paper, we address this drawback by providing a framework to actively choose between superposed states whenever a player observes something new. We do this by framing strong story experience managers as adversarial game players to show that some states are more desirable than others based on the percentage of their child branches that are consistent with authorial constraints. We then provide a model capable of representing, updating, and splitting superposed state information. This model can be used to actively choose states that maximize the chance of authorial constraints holding whenever the user makes an observation that collapses the superposition.

Experience Management Framework

This paper builds off of work on automated strong story interactive narrative systems. A common approach to realizing these systems is AI planning (Young et al. 2013). Our work is implemented in the framework of a turn-based, state-centric experience management framework sim-

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Problem

Initial State
Snake at Elevator-Room
Snake has PP7
Snake has C4
Snake has C4-Detonator
Boss at Gear-Room
Boss has Laser-Rifle
Boss has Trip-Wire
Boss has Phone
Gears at Gear-Room
Terminal1 at Gear-Room
Terminal2 at Left-Walkway
Terminal3 at Right-Walkway
Terminal4 at Platform
Elevator-Room connected Gear-Room
Gear-Room connected Left-Walkway
Gear-Room connected Right-Walkway
Left-Walkway connected Platform
Right-Walkway connected Platform

Goal State
Snake is alive
Boss is not alive
Boss is at Platform
Phone is linked
Snake disabled a trap
Gears are destroyed

Domain

move(?mover,?loc,?oldloc)
Precons: ?mover at ?oldloc
?mover is alive
?oldloc connected ?loc
Effects: ?mover not at ?oldloc
?mover at ?loc

use(?user,?computer,?loc)
Precons: ?user at ?loc
?user is alive
?computer at ?loc
Effects: ?computer is used

place(?actr,?bmb,?thng,?loc)
Precons: ?actr at ?loc
?actr is alive
?actr has ?bmb
?thng at ?loc
Effects: ?actr not have ?bmb
?bmb placed on ?thng

shoot(?shooter,?shot,?gun,?loc)
Precons: ?shooter at ?loc
?shooter is alive
?shooter has ?gun
?shot at ?loc
Effect: ?shot is not alive

make(?maker,?gun,?wire)
Precons: ?maker has ?gun
?maker has ?wire
Effects: ?maker has Trap
?maker not have ?wire
?maker not have ?gun

detonate(?actr,?dtn,?bmb,?thng)
Precons: ?actr has ?dtn
?actr is alive
?bmb placed on ?thng
Effects: ?thng is destroyed
?bmb not on ?thng

disable(?disabler,?trap,?loc)
Precons: ?disabler at ?loc
?disabler is alive
?trap at ?loc
Effects: ?trap not at ?loc
?disabler disabled ?trap

set(?setter,?trap,?loc)
Precons: ?user at ?loc
?user has ?trap
?user is alive
Effects: ?trap is at ?loc

link(?user,?phone,?cmptr,?loc)
Precons: ?user at ?loc
?user is alive
?user has ?phone
?cmptr is used
?cmptr at ?loc
Effects: ?phone is linked

Figure 1: A simplified PDDL problem and domain that represents the *Spy* world.

(a) The Boss (b) Snake (c) PC Terminal (d) Activated PC (e) Phone (f) Linked Phone (g) Gears (h) Trap

Figure 2: A key of PDDL objects pictured in Figures 3, 4, and 6.

ilar to the GME system (Robertson and Young 2014b; 2015). The framework utilizes the Planning Domain Definition Language (PDDL) (McDermott et al. 1998) to model world states and dynamics. PDDL models are comprised of an initial state, a goal state, and a set of action operators that characters use to transform the story world. Figure 1 shows an example PDDL model of the *Spy* domain.

PDDL representations consist of a problem and a domain. A PDDL problem containts an initial state that specifies what things are true when the world begins. A PDDL problem also contains a goal state that specifies what things should be true when the story ends. A PDDL domain specifies action operators that can be used by characters to update states. Action operators have a list of preconditions, which are terms that must be true for the action to be executed. Action operators also have a list of effects, which are terms that become true after the action is executed. Finally, action operators have a list of variables. In Figure 1, variables are denoted with a leading '?'. For example, *?mover* in the *move* action is a variable because its first character is a '?'.

Variables can be bound to certain objects, which are things that exist in the world. In Figure 1, objects are specified with a capitalized first letter. For example, *Snake* is an object because its first character is a capital 'S'. When all the variables in an action operator are bound to objects, it is called *fully ground*. Fully ground operators are actions that can be performed by characters in the story world. For example, when the action *move(?mover,?loc,?oldloc)* is bound to the objects *Snake,Gear-Room,Elevator-Room* it represents

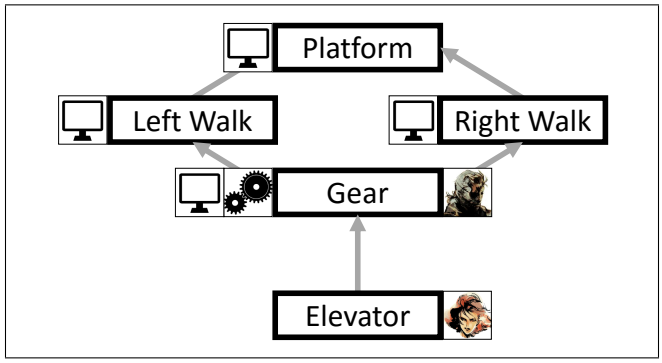
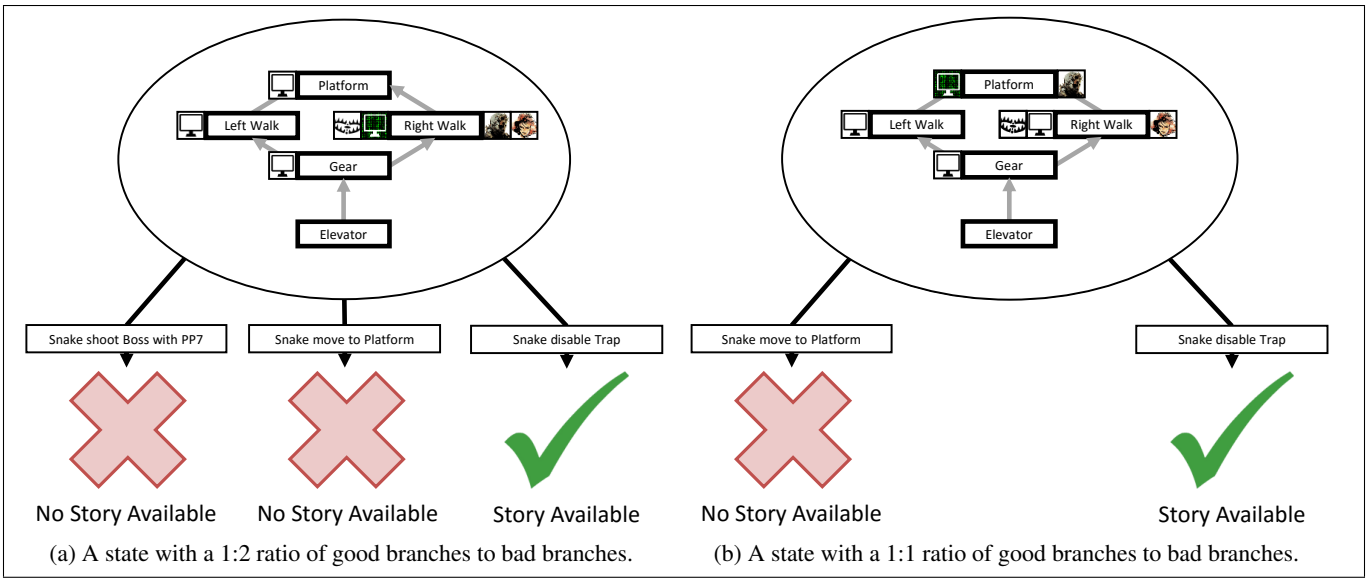


Figure 3: A diagram of the initial configuration of the *Spy* world.

the character Snake moving from a place called the Elevator Room to a place called the Gear Room. This action can be performed in any world state where the action's bound preconditions are satisfied: where Snake is located at the Elevator Room, Snake is alive, and the Elevator Room is connected to the Gear Room. The new state created by applying this action, called the *successor state*, will have Snake located at the Gear Room instead of the Elevator Room.

The *Spy* game, modelled by the PDDL domain and problem in Figure 1, is an example world where the player, as a spy named Snake, must foil the final attempt of the computer-controlled antagonist, the Boss, to bring a weaponized satellite online. The confrontation takes place on a sattellite dish antenna cradle with five discrete loca-



(a) A state with a 1:2 ratio of good branches to bad branches.

(b) A state with a 1:1 ratio of good branches to bad branches.

Figure 4: Two states and their outgoing player actions in the *Spy* domain. Assuming the player makes uniformly random decisions, the state in Figure 5a has a 1/3 chance that authorial constraints hold and Figure 5b has a 1/2 chance.

tions where the Snake and Boss can 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 layout is pictured in Figure 3, where locations are labeled rectangles and doors are arrows. The doors can only be traversed in the direction arrows are facing. Snake begins the game in the Elevator Room. Her job is to disable the satellite dish's alignment mechanism in the Gear Room and eliminate the Boss. The Boss is trying to send instructions from his phone to the satellite by linking the phone to one of four computer terminals on the cradle. The domain author wants the Boss to build and set a trap to be disabled by the player before a final confrontation between the two on the platform. These authorial constraints are coded as conditions in the PDDL goal state.

Snake starts off in the elevator room and the Boss begins in the gear room. Snake has a pistol (PP7) an explosive (C4) and a detonator for the explosive. The Boss has a laser rifle and a trip-wire for building a trap, and a phone. There is a computer terminal on the Platform, Left and Right walkways, and in the Gear Room. This initial world configuration is shown in Figure 3. The game progresses by alternating between allowing the Boss and Snake to take an action that updates state information. The Boss is controlled by plots generated by the system's planner. Snake is controlled by a player. The game continues until a goal state is reached or the author's constraints are broken.

Experience Management as Adversarial Game

One way to view experience management is as an adversariable game played by the experience manager. The experience manager wins if it tells an interesting story and loses if it tells an uninteresting story. Viewing experience management from this adversarial game perspective has been around since the Oz Project (Weyhrauch 1997; Mateas 2001). This perspective is useful because the win and loss outcomes can quantify how succesful an algorithm is at telling interesting interactive stories. It also serves as a baseline that allows us to compare the output of different experience management algorithms.

Interesting story qualities like intentional character actions (Riedl and Young 2010; Haslum 2012), character beliefs (Teutenberg and Porteous 2015), and conflict (Ware and Young 2014) can be modeled with PDDL, reasoned about by planners, and exist in solution plans. If an author can compile all the narrative qualities they care about into a PDDL domain and problem, the experience management framework outlined in the last section can be viewed as an adversarial game player. The experience manager wins the game by telling an interesting story if a goal state is reached. It loses the game by telling an uninteresting story if all goal states become inaccessible.

State Utility

Under the adversarial game perspective, not all states part of valid plots are equally desirable. The utility of any state can be measured with the probability that the world will reach a goal configuration. Unfortunately, this probability is hard to determine for several reasons. One difference between experience management and a more traditional game like Chess is that in most games the objectives and outcomes of both players are explicitly defined and often symmetric. In an interactive narrative domain we don't always know the player's payoffs or how they will act. With a model of choice preference (Yu and Riedl 2013), goal recognition (Cardona-Rivera and Young 2015), and/or role assignment (Domınguez et al. 2016) we could favor player choices predicted by the player model with higher probability. For now, we assume that the player will choose uniformly at random from the available choice options.

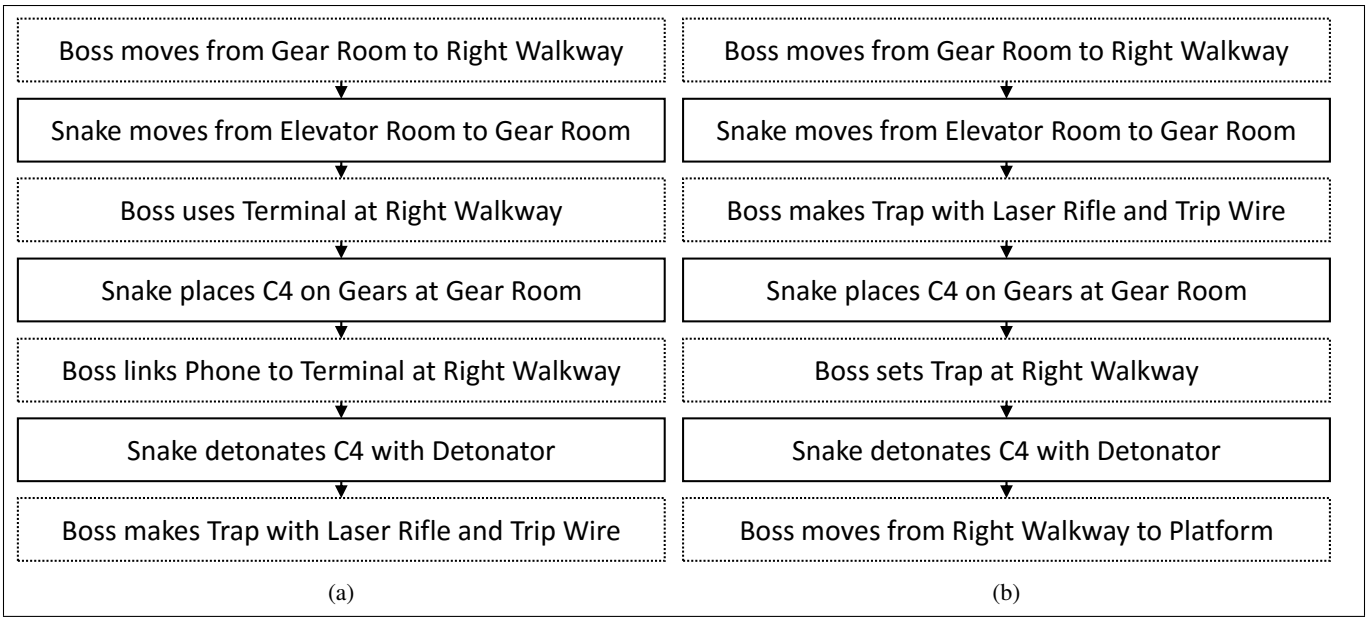


Figure 5: Two perceptually equivalent action trajectories in the *Spy* domain. The player observes any action performed in the room where they are located. Actions unobserved by the player have a dotted border. From the player's perspective, either of these sequences of events could have taken place given player actions of moving to the Gear Room, placing C4 on the gears, and detonating the C4.

Under this assumption, a state's utility can be calculated by fully expanding all of its outgoing edges and counting the ratio of wins to losses. An example is given in Figure 4. Both states are part of possible story sequences of comparable length and both are within three actions of a goal configuration. In Figure 4a, the Boss and Snake are at the Right Walkway. The Boss has crafted and laid a trap for Snake, has activated the computer terminal, and linked his phone. In Figure 4b the Boss is at the Platform and Snake is at the Right Walkway. The Boss has laid a trap for Snake at the Right Walkway, but instead of activating and linking his phone at the Right Walkway, has moved to the Platform and activated the computer terminal there.

It is the player's turn in both states. The outgoing edges represent actions available to the player. In Figure 4a, the player as Snake can shoot the Boss, move to the Platform, or disable the trap set by the Boss. The first two actions break authorial constraints. If the player shoots the Boss, the Boss cannot be at the Platform at the end of the story. If the player moves to the Platform, they cannot disable the trap set at the Right Walkway. If the player disables the trap, Boss escapes to the platform where all the authorial constraints can be fulfilled. So, assuming the player acts randomly, the experience manager has a 1 in 3 chance of producing a story that fulfills authorial constraints from the state in Figure 4a.

In Figure 4b, Boss has moved to the Platform so the player can no longer shoot the Boss. This takes away one of the branches where the experience manager loses. It now has a 1 in 2 chance of producing a story that fulfills authorial constraints. If given a choice between these two states, an experience manager should choose the state in Figure 4b because there is a higher chance that a story matching authorial constraints will play out than the state in Figure 4a.

Choosing States

Experience managers can take advantage of state utility to maximize the probability of telling a story where author constraints are satisfied. A process called event revision (Robertson and Young 2014a) searches through alternate histories consistent with player observations when looking for stories that match authorial constraints. For example, the two stories shown in Figure 5 are perceptually equivalent from the player's perspective. If an experience manager decided to switch from one of these world histories to another, the player wouldn't know the difference. One downside of this approach is it waits until the player acts out of alignment with the current story model to conduct search. A better method would be to proactively choose between alternate state models based on utility when a player learns something new about the story world.

These alternate possible, perceptually equivalent histories form a collection, or superposition, of states the player could exist in. Figure 6 shows a collection of six superposed states that correspond to six perceptually equivalent world histories. The set of world histories include the two shown in Figure 5. Whenever an NPC has the option of changing something in the world without the player observing, the superposition grows. Whenever the player observes something new about the world, the superposition is split. Whenever a superposition is split, the experience manager transitions the player into one of the newly split states. Currently, this is a passive transition based on the experience manager's current story. If the experience manager tracked these superposed states and evaluated their utility, it could transition the player to the split state that maximizes the probability that an interesting story will be told every time the player learns something new about the world.

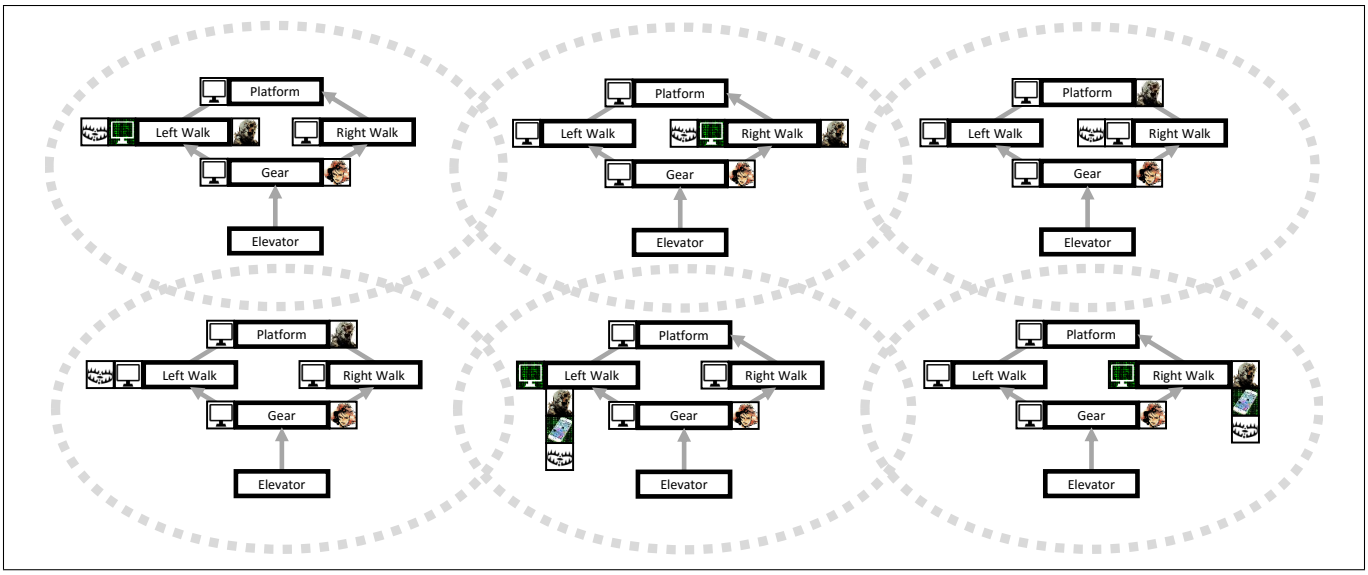


Figure 6: A set of six superposed states, each consistent with what the player knows about the world in the *Spy* domain after they perform the actions: move from Elevator to Gear Room, place C4 on Gears, detonate C4. Two of these states are produced by the stories in Figure 5.

For example, if the player chooses to move to the Right Walkway from any of the superposed states pictured in Figure 6, they will observe everything located at the Right Walkway. This observation will split the superposition into four parts, of which the player will exist in one. The experience manager could show the player that only a computer terminal exists at the Right Walkway. If this is the case, the player exists in a new superposition consisting of the three states where only an unused computer terminal is at the Right Walkway. Each of the other three states form their own seperate possibility. The player could observe the Boss, a used computer terminal, and a set trap at the Right Walkway. Or the player could observe a set trap and an unused terminal at the Right Walkway. Or the player could observe Boss holding a trap and phone linked to a computer terminal.

These four possibilities form a choice for the experience manager immediately before the player makes a new observation. If the experience manager can choose the split state with the highest utility, it can maximize the probability that a goal state will be reached. There are two major hurdles that must be overcome to make this happen. First, the set of all possible worlds that are consistent with a player's observations must be tracked and updated. Second, the utility of these states must be calculated or estimated in order for the system to make intelligent decisions.

Superposition Model

The rest of this paper focuses on modelling and updating the set of superposed states consistent with player knowledge. To make decisions based on the utility of superposed state, the system must first be able to enumerate the set of states consistent with player knowledge. One way to enumerate the set of possible states is to model each state seperately and update each state individually as play progresses. One problem with this approach is that not only will the set of states grow quickly as NPCs take unobserved actions, but the time it takes to update the set grows faster. To update the set, each possible action the current NPC could perform on each state must be applied to create the successor superposition. To mitigate this cost, we present a method of modelling all states in a single data structure that can be updated by applying all possible NPC actions once.

Modeling Superposed Formulae

Similar to a process called Initial State Revision (Riedl and Young 2005), we model superposed states as a single data structure where formula can be true, false, or undetermined. A formula is true or false when it is known by the player. A formula is undetermined when there exists a possible world consistent with the player's observations where the formula is true and also one where it is false. For example, if it is consistent with the player's knowledge for the Boss to either be at the Gear Room or the Right Walkway, the formula that represents the Boss being located at the Gear Room, *(at boss gear)*, would be in the undetermined category. It would be in this category because it is consistent with the player's observations for the formula to be either true or false.

Creating the Superposition

To create a state superposition, the experience manager must first calculate all unobserved actions that could be performed by the current NPC in the current state. If the effect of any of these actions reversed a state formula, the formula is moved from true or false to unknown in the successor superposition. For example, one thing the Boss can do from the initial state is move from the Gear Room to the Right Walkway. This action would not be observed by the player and it reverses the formula *(at boss gear)* from true to false and *(at boss right)* from false to true. Both of these formulae would be moved to the unknown category in the successor superposition.

(used terminal1 boss)		(not (used terminal1 boss))	
TRUE	**FALSE**	**TRUE**	**FALSE**
(at boss gear)	(has boss trap)		
(has boss wire)	(at boss left)		
(has boss rifle)	(at boss right)		

(at boss left)		(not (at boss left))	
TRUE	**FALSE**	**TRUE**	**FALSE**
(has boss wire)	(at boss gear)		
(has boss rifle)	(has boss trap)		
	(at boss right)		
	(used terminal1 boss)		

(has boss wire)		(not (has boss wire))	
TRUE	**FALSE**	**TRUE**	**FALSE**
(has boss rifle)	(has boss trap)	(at boss gear)	(at boss left)
		(has boss trap)	(at boss right)
			(has boss rifle)
			(used terminal1 boss)

(has boss trap)		(not (has boss trap))	
TRUE	**FALSE**	**TRUE**	**FALSE**
(at boss gear)	(has boss wire)	(has boss wire)	
	(at boss left)	(has boss rifle)	
	(at boss right)		
	(has boss rifle)		
	(used terminal1 boss)		

(has boss rifle)		(not (has boss rifle))	
TRUE	**FALSE**	**TRUE**	**FALSE**
(has boss wire)	(has boss trap)	(at boss gear)	(has boss wire)
		(has boss trap)	(at boss left)
			(at boss right)
			(used terminal1 boss)

(at boss gear)		(not (at boss gear))	
TRUE	**FALSE**	**TRUE**	**FALSE**
	(at boss left)	(has boss wire)	(has boss trap)
	(at boss right)	(has boss rifle)	(used terminal1 boss)

(at boss right)		(not (at boss right))	
TRUE	**FALSE**	**TRUE**	**FALSE**
(has boss wire)	(at boss gear)		
(has boss rifle)	(has boss trap)		
	(at boss left)		
	(used terminal1 boss)		

Figure 7: Dependencies in the *Spy* world after one turn of the Boss acting without being observed. Each formula in the superposed state is listed as true in the left column and false in the right column. Underneath each formula is a list of what must be true and what must be false if the superposed state is split in that direction.

Modeling Superposition Dependencies

In order to split a superposition, dependency information must be tracked between unknown formulae. For example, if the player were to move from the Elevator Room to the Gear Room on their first move, the system would need to make a decision about whether *(at boss gear)* was true or false. If the system decided that the formula was true, it would need to know that *(at boss right)* should become false, since the Boss cannot be in two places at once. One way to model this information is with dependencies similar to Graphplan's (Blum and Furst 1997) mutex links, but applied to true/true and false/false relationships as well as true/false relationships.

Here is a method to calculate these dependencies: for each unknown formula A, cycle through every other unknown formula B. If in all states where A is true or unknown, B is true, draw a true/true link from A to B. If B is false in all states where A is true or unknown, draw a true/false link from A to B. If in all states where A is false or unknown, B is true, draw a false/true link from A to B. If B is false in all states where A is false or unknown, draw a false/false link from A to B. If none of these conditions apply, B is independent of A and no links are drawn.

The output of applying this method to the set of actions available to the Boss in the initial state is given in Figure 7. Outgoing true and false links are underneath true formulae in the left column and false formulae in the right column. For example, if the system decides *(has boss trap)* is true, the only way for this to happen is if the Boss used his first turn to craft the trap from his rifle and trip wire. This means the Boss couldn't have moved from the gear room, so *(at boss gear)* must be true and *(at boss left)* and *(at boss right)* must be false. The Boss also could not have turned the computer terminal at the gear room on, so *(used terminal1 boss)* must be false. And since the only way to make a trap is to use a rifle and wire, *(has boss rifle)* and *(has boss wire)* are false.

It is important to note that this model is correct only if the domain allows characters to take no action during their turn. Otherwise, it will not always model all dependencies.

Updating and Splitting

Once a superposition is created, it can be updated by applying new actions. When applying new actions, unknown formulae can fulfill both true and false preconditions. For example, when determining if *(move boss platform right)* can be performed from the superposition pictured in Figure 7, *(at boss right)* is in the superposed unknown category so it fulfills the true precondition *(at boss right)*.

When a superposition is split by a player observation, the system must decide whether the observed formula becomes true or false. When this happens, all linked dependencies must also become true or false. For example, if the system decides that *(at boss gear)* is true when the player moves from the elevator room to the gear room, it must also make *(at boss left)* and *(at boss right)* false.

Future Work

This model of superposed states consistent with player knowledge is only half the information needed to make decisions about what states to choose as the player learns about the story world. The other half is utility information that specifies what states are better than others. The system currently has to fully expand the branches underneath each possible state to find utility information. However, fully expanding branches will not be computationally feasible in most cases. The next step for this work will be to create an effective way to gather utility information without solving the game tree under each state.

Conclusion

In this paper we view experience management as an adversarial search problem and present a concise way to model multiple states consistent with a player's knowledge as they play. This model can be applied to maximize the probability author constraints are upheld as story events play out.

References

Blum, A. L., and Furst, M. L. 1997. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence* 90(1):281–300.

Bruckman, A. 1990. The Combinatorics of Storytelling: Mystery Train Interactive. Master's thesis, The MIT Media Laboratory.

Cardona-Rivera, R. E., and Young, R. M. 2015. Symbolic Plan Recognition in Interactive Narrative Environments. In *The Eight Intelligent Narrative Technologies Workshop at AIIDE*.

Domınguez, I. X.; Cardona-Rivera, R. E.; Vance, J. K.; and Roberts, D. L. 2016. The Mimesis Effect: The Effect of Roles on Player Choice in Interactive Narrative Role-Playing Games. In *Proceedings of the 34th Annual CHI Conference on Human Factors in Computing Systems*.

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

Louchart, S., and Aylett, R. 2003. Solving the Narrative Paradox in VEs – Lessons from RPGs. In *Intelligent Virtual Agents*, 244–248.

Magerko, B. 2007. Evaluating Preemptive Story Direction in the Interactive Drama Architecture. *Journal of Game Development* 2(3):25–52.

Mateas, M. 2001. An Oz-Centric Review of Interactive Drama and Believable Agents. *Artificial Intelligence Today* 297–328.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language. Technical Report CVC TR98003/DCSTR1165, Yale Center for Computational Vision and Control.

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

Riedl, M., and Bulitko, V. 2013. Interactive Narrative: An Intelligent Systems Approach. *AI Magazine* 34(1):67–77.

Riedl, M. O., and Young, R. M. 2005. Open-World Planning for Story Generation. In *International Joint Conference on Artificial Intelligence*.

Riedl, M. O., and Young, R. M. 2010. Narrative Planning: Balancing Plot and Character. *Journal of Artificial Intelligence Research* 39(1):217–268.

Riedl, M. O.; Stern, A.; Dini, D. M.; and Alderman, J. M. 2008. Dynamic Experience Management in Virtual Worlds for Entertainment, Education, and Training. *International Transactions on Systems Science and Applications* 4(2):23–42.

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 International Conference on Autonomous Agents and Multiagent Systems*, 741–748.

Riedl, M.; Thue, D.; and Bulitko, V. 2011. Game AI as Storytelling. In *Artificial Intelligence for Computer Games*. Springer. 125–150.

Roberts, D. L., and Isbell, C. L. 2008. A Survey and Qualitative Analysis of Recent Advances in Drama Management. *International Transactions on Systems Science and Applications, Special Issue on Agent Based Systems for Human Learning* 4(2):61–75.

Robertson, J., and Young, R. M. 2014a. Finding Schrödinger's Gun. In *Artificial Intelligence and Interactive Digital Entertainment*, 153–159.

Robertson, J., and Young, R. M. 2014b. Gameplay as On-Line Mediation Search. In *The First Experimental AI in Games at the Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Robertson, J., and Young, R. M. 2015. Automated Gameplay Generation from Declarative World Representations. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Teutenberg, J., and Porteous, J. 2015. Incorporating Global and Local Knowledge in Intentional Narrative Planning. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, 1539–1546.

Ware, S. G., and Young, R. M. 2014. Glaive: A State-Space Narrative Planner Supporting Intentionality and Conflict. In *Tenth Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Weyhrauch, P. 1997. *Guiding Interactive Drama*. Ph.D. Dissertation, Carnegie Mellon University Pittsburgh, PA. CMU-CS-97-109.

Young, R. M.; Ware, S. G.; Cassell, B. A.; and Robertson, J. 2013. Plans and Planning in Narrative Generation: A Review of Plan-Based Approaches to the Generation of Story, Discourse and Interactivity in Narratives. *SDV. Sprache und Datenverarbeitung*.

Yu, H., and Riedl, M. O. 2013. Data-Driven Personalized Drama Management. In *Ninth Conference on Artificial Intelligence for Interactive Digital Entertainment*, 191–197.