Experimental AI in Games: Papers from the AIIDE Workshop
AAAI Technical Report WS-16-22

Dear Leader's Happy Story Time: A Party Game Based on Automated Story Generation

Ian D. Horswill
Northwestern University
Evanston, IL
ian@northwestern.edu

Abstract

Players in *Dear Leader's Happy Story Time* are placed in the role of contestants in a reality TV show where they are forced to audition for roles in the upcoming film of the host, a deranged billionaire who has inexplicably been elected president. The stories are produced by a story generator that combines stock plots and characters to produce kitsch story outlines. The players then collaborate to improvise a camp performance of the outline. The game design provides a context for experimenting with automatic story generation within a narrative game, as well as an opportunity for experimenting with knowledge representation schemes for expressing the tropes of popular narrative. The story generator uses a higher-order logic for describing tropes, and an HTN planning algorithm based on Nau *et al.*'s SHOP.

Introduction

One of the issues with using story generation in games is that current generators cannot compete with human-constructed narratives, and so cannot easily be used in conventional game genres. Players would be unlikely to accept an entry in the *Mass Effect* franchise (BioWare 2012) in which characters spoke lines generated by current story generation technology. This does not, however, mean that story generators cannot be used to interesting effect in games; merely that the game design must be altered to better match the strengths and weaknesses of current technology. One promising approach is to design games in which the system shares authorship with the players in such a way as to make best use of the strengths of each party (O'Neill et al. 2011; Ryan, Samuel, and Summerville 2016; Reed and Garbe 2016; Horswill 2015; Swanson and Gordon 2012).

In this paper, we describe work in progress on *Dear Leader's Happy Story Time*, a party game in which players act out an automatically generated story as the beats are presented to them in real time. The stories are as bad as one would expect from an automated story generator. But it works as a party game because the players can deliberately camp up their performances.

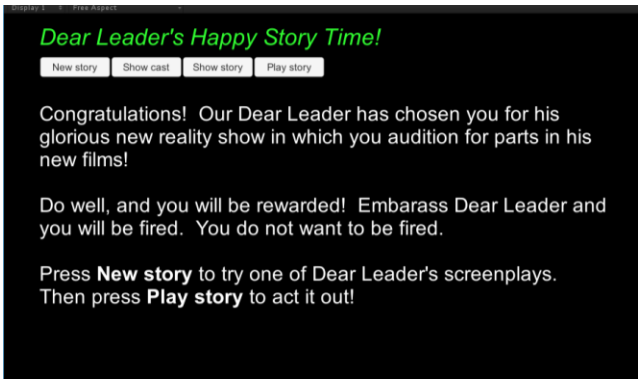

Figure 1: Screenshot from game

Dear Leader's Happy Story Time

The game is set in an unlikely dystopian future in which a billionaire and former reality TV star, referred to only as "Dear Leader," declares upon his election as president that he always wanted to direct. He starts a new reality show, *Dear Leader's Happy Story Time*, in which convicts are given the opportunity to earn their freedom by auditioning for places in his next film. Unfortunately, Dear Leader is as good a screenwriter as Hitler was a painter; and his tastes tend toward "wholesome, American" (read: kitsch) stories. Worse, the scripts are only outlines, specifying the general content of beats, such as "the lovers share a tearful reunion," forcing the players to improvise as they are read the beats of the plot, one at a time, without knowing what beat is coming

next. At the end, Dear Leader, along with the viewing public, determine which contestants are hired, which are fired (lethally), and which will live to play another day.

Computationally, the game consists of a story generator together with a collection of stock characters, settings, and plot tropes. The story generator is essentially an HTN planner, with plot tropes represented as methods for the planner (see Story Language, below). The generator finds a randomized solution to its top-level task, `story`, then presents the beats of the plan, one at a time, to the players as they act out the story. A prototype of the game has been developed under *Unity3D* (Unity Technologies 2004).

Aesthetics

Although there have been a number of *avant-garde* experiments with automated story generation, such as NaNoGenMo (Kazemi 2013) and even a published novel, *World Clock* (Montfort 2013), current story generators are extremely limited. They do not produce stories that typical audiences would choose to consume for entertainment. This places a serious constraint on the use of story generators in games intended for such audiences: the context of use of the generator must be such that the audience will consider the generated stories satisfying.

The approach taken in *Dear Leader* is to salvage bad stories through playful, ironic performance; calling attention to their artificiality and theatricality; and surfacing the mechanistic character of their generation. The story generator, which is effectively generating random combinations of stock tropes, cannot fail to be kitsch in Greenberg's technical sense (Greenberg 1939). So rather than trying to mitigate the kitsch, we try to exaggerate it, creating stories that seem like TV movies made for the Lifetime Channel by an alien intelligence that had read, but not understood, the TV Tropes wiki (TV Tropes community 2004). We then encourage the players to perform these stories with equal exaggeration.

Ideally, this exaggeration will allow the stories to be experienced as camp. However, as Sontag argues (2013), something can't be pure camp if it self-consciously seeks to be camp, so perhaps the best we can hope for is post-modern irony. In any case, we hope that with repeated play, the reflexive nature of the gameplay and the surfacing of the mechanics of story creation will create a variant of Wardrip-Fruin's (2011) SimCity effect, inviting the player to reflect on the structures of popular narrative in much the same way as reading TV Tropes.

Story language

This project began as an attempt to represent the tropes of popular narrative, particularly genre screenwriting in the Hollywood tradition as codified on TV Tropes (TV Tropes community 2004). Consequently, the technical focus in this project has been on making expressive languages for representing plot tropes, rather than on powerful planning algorithms. As such, the language used for representing plot tropes can be thought of as a kind of domain-specific language for representing story fragments, together with constraints on their application. It can also be thought of as a fancy story grammar (Mandler and Johnson 1977), albeit not a context-free one, since it makes liberal use not only of unification, but of planning and inference.

A finished story in *Dear Leader* is represented both as a set of *plot events*, and as a *story world* in which those events take place. The plot events are generated by a total-order, hierarchical task network (HTN) planner.

The story world is similar to *Fiasco*'s notion of a "setup" (Morningstar 2009); it includes information such as what characters, settings, and themes appear in the story, who is the protagonist, which characters are sympathetic, etc. It is distinct from the world *state* tracked by the planner; state can be changed by actions, but story world information is fixed over the course of the story (i.e. the story world contains no fluents).

The story world is represented as a set of ground literals in a higher-order logic. The programmer can declare facts about symmetries of predicates, or that one predicate generalizes another. The programmer can also declare inference rules within the logic, and that given pairs of literals are contradictory. The story generator guarantees that the generated story world is contradiction-free given the provided inference and contradiction rules.

Declarations of plot tropes and beats can contain constraints on the story world. The story planner will satisfy them as best it can, backtracking when a plot event's constraint cannot be satisfied in the current story world.

The system's plot knowledge consists primarily of a set of rules for the HTN planner that map plot events to sequences of lower-level plot events. Because of the narrative domain, we refer to HTN tasks as *plot events* rather than tasks. In particular, we will refer to the primitive operators as *beats*, and the methods as *plot tropes*. The system's knowledge is therefore contained in large part in its stock of beats and plot tropes.

A beat declaration specifies the name and arguments of the beat, together with a template for generating English text to describe the beat:

```
beat(distraught(Person),
     $text("[Person] is distraught")).
```

Here we follow the Prolog/ASP convention of denoting logic variables through capitalization. Beat declarations may also optionally include preconditions on state, as well

as add lists, and delete lists to modify the state. For example:

```
beat(distraught(Person) :
         { adds(distraught(Person)) },
     $text("[Person] is distraught")).
```

They may also include constraints on the story world (again, as distinct from state), either in the form of domain constraints for the beat's arguments, as in:

```
beat(kills(Killer, Victim,
           Instrument: murder_weapon),
     $text("[Killer] kills [Victim]
            with [Instrument]")).
```

which states that the Instrument must be a murder weapon, or as explicit conjunction of literals:

```
beat(kills(Killer, Victim,
           Instrument: murder_weapon)
     : { constraint:
         (unsympathetic(Killer),
          Killer \= Victim) },
     $text("[Killer] kills [Victim]
            with [Instrument]")).
```

Plot trope declarations are more or less the same in the sense that they specify the name and arguments for a plot event together with an expansion for it. However, in this case the expansion is a series of other plot events rather than English text. Like beat declarations, they can include constraints on the story world, preconditions, etc. For example, the plot trope:

```
introduce_relationship(P : protagonist,
                       L : character,
                       lovers(P, L)) :
      (P \= L, compatibility(P, L, C))
   ==>
      meet(P, L, _),
      fall_in_love(P, L, C).
```

states that you can introduce the relationship of two characters, P (the protagonist) and L, being lovers by having them meet someplace and fall in love. However, within the story world, P and L must be different characters, and they must share some romantic compatibility C. Note that arguments to plot events may be literals or other plot events, not just objects from the story world. This allows higher-order constructs such as the task of achieving some condition to be expressed.

Example trope

"Boy meets girl" is a popular trope in narrative. We will adopt the alternate name "peep meets peep" to remain gender neutral. In many of its instantiations, the trope involves two characters meeting and falling in love, then breaking up, and finally reuniting. We can therefore represent it with a rule such as:

```
peep_meets_peep(P, L) ==>
   fall_in_love(P, L, Cof),
   breakup(P, L),
   reunite(P, L).
```

where P is the protagonist and L their love interest. Here we assume that the plot events `fall_in_love`, `breakup`, and `reunite` would each have their own trope rules for decomposing them into simpler plot events, and eventually into beats.

However, this trope is a subtrope of a more general trope involving loss and redemption. In the parent trope, the protagonist first loses some property or relation, and then reestablishes it. We can represent this more general trope using the rule:

```
loss_and_redemption ==>
   introduce_relationship(P, O, R),
   break_relationship(P, O, R),
   restore_relationship(P, O, R).
```

stating that we first introduce some relationship R between the protagonist P and the other O, then break it, then restore it. The peep meets peep trope is then simply a special case where $R = \text{lovers}(P, Q)$. The `introduce_relationship` rule in the previous section shows an example for introducing the `lovers` relationship. For reasons of space, we'll omit the rules for breaking the relationship and focus on restoring the relationship. We start with the rules:

```
restore_relationship(P, L,
                     lovers(P, L)) ==>
   saves_life(P, L),
   profess_love(L, P).
restore_relationship(P, L,
                     lovers(P, L)) ==>
   saves_life(L, P),
   profess_love(L, P).
restore_relationship(P, L,
                     lovers(P, L)) ==>
   attempt_suicide(P, _),
   prevents_suicide(L, P),
   profess_love(L, P).
```

These rules state that you can reunite lovers by having one save the life of another or by having the rejected protagonist attempt suicide only to be saved by their love interest. In all

cases, the close call with death forces the love interest to realize the depth of their love for the protagonist.

Saving the life of a character can be described by:

```
saves_life(Saver, Savee : character) :
   threatening_situation(Savee, Threat)
==>
   introduce_threat(Savee, Threat),
   rescue(Savee, Saver, Threat).
```

Threatening situations can include things like being mugged, or medical emergencies affecting either the Savee or someone beloved to them, such as a parent, sibling, or pet ferret:

```
threatening_situation(_, mugger(_)).
threatening_situation(P,
          medical_situation(P, M)) :-
   medical_situation(M).
threatening_situation(P,
          medical_situation(Who, M)) :-
   beloved_of(P, Who),
   medical_situation(M).
```

In the case of a medical situation, we introduce the threat by first telling the audience the specific Patient is threatened, then letting them know that the Subject (ultimately, the love interest who rejected the protagonist) is distraught:[1]

```
introduce_threat(
      Subject,
      medical_situation(Patient, What))
  ==>
   life_threatened_by(Patient, What),
   when(Patient \= Subject,
        distraught(Subject)).
```

The protagonist can then save whoever is ill by donating a kidney or some other valuable medical item:

```
rescue(Rescued, Saver,
       medical_situation(Who, _))
   : (medical_donation(Donation),
      character(Who))
 ==>
   setting(hospital),
   gives(Saver, Who, Donation),
   when(Rescued \= Who,
        tearful_reunion(Rescued, Who)).
```

This is a general trope that would be applicable to any Saver character. Additional rules could be added for specific characters or types of characters: doctors can save the patient directly; supernatural characters might save them through magic.

The above rules collectively define the loss-and-redemption trope for the `lovers` relationship. Others can extend it to allow the breakup and reestablishment of a marriage, or a friendship (as in a buddy movie). Or we could further abstract it to accommodate more general predicates such as the loss and redemption of a character's honor.

We can also provide rules for subverting the trope, for example by providing a *yandere*[2] plot line in which, rather than reestablishing the relationship, the jilted lover stalks the protagonist:

```
yandere_plot(P, S : character)
  : (P \= S,
     dif(Tone1, Tone2, Tone3))
  ==>
   introduce_relationship
              (P, S, lovers(P, S)),
   break_relationship(P, S,
                      lovers(P, S)),
   stalks(S, P, Tone1),
   stalks(S, P, Tone2),
   stalks(S, P, Tone3),
   stalker_confrontation(P, S).
```

This begins identically to the peep-meet-peep trope, but ends with a series of stalking incidents (the `dif` predicate requires they all have different "tones"), and finally a confrontation.

Story generator implementation

The story generator is ultimately a forward-search, ordered, HTN planner, similar to SHOP (Nau et al. 1999). The planner tracks story state as it plans, testing candidate plot events for preconditions, and updating the state based on add and delete lists, in essentially the same manner as SHOP.

Although the details are outside the scope of this paper, the SHOP algorithm can be seen as an extension of the "threading" technique used in definite clause grammars (Pereira and Warren 1980), wherein grammar rules are translated directly into Prolog rules for predicates with extra arguments (Sterling and Shapiro 1994). We use this technique to macro-expand beat and trope declarations directly into Prolog rules that can be executed directly. In particular, a plot event $e(a_1, \ldots, a_n)$ is expanded into a Prolog predicate $e(a_1, \ldots, a_n, s_{in}, s_{out}, p_{in}, p_{out})$ that is true whenever p_{in} is a plan that enacts $e(a_1, \ldots, a_n)$, and that, when run in state

[1] `when(`P`,` T`)` expands to T when P is true, otherwise to the empty plan.

[2] A *yandere* is a manga/anime trope involving love interests who are violent sociopaths. While typically female in those genres, *Dear Leader* is gender neutral.

s_{in}, results in state s_{out}.[3] A Prolog query of the predicate will cause Prolog to do a forward state-space search for a plan that solves the task $e(a_1, \ldots, a_n)$. This technique allows planning to be relatively fast in spite of the use of an interpreted Prolog. In addition, it allows the injection of arbitrary Prolog code during the macro-expansion process. Finally, it allows arguments to task to be not just objects, but literals in the underlying logic, such as in the examples above. This makes the language much more expressive.

Story world constraints specified in a rule are added as subgoals at the beginning of the Prolog rule. The story planner maintains a model of the story world as it generates the story. When a plot rule stipulates a constraint, it compares it to the existing model, checking for contradictions. If none are found, it adds it to the model, along with all its implications given the current model, to deductive closure. If any contradictions are found, it backtracks, rejecting the current plot rule.

The Prolog interpreter allows specific predicates to be declared randomizable, in which case it will shuffle the order in which it tries the clauses for that predicate, making it straightforward to generate randomized stories.

Text generation is performed using a variant of definite clause grammars (Pereira and Warren 1980). Each beat declaration includes a DCG for generating text describing the beat, and additional DCG rules can be included to describe ancillary generation processes such as how to realize particular entities as noun phrases.

Example story

Here is an example story generated by *Dear Leader* using the tropes discussed above. It tells a love story involving the manic pixie dream peep and a vampire noble. The text and formatting are as they appear in-game:

Setting: Starbuck's
The manic pixie dream peep meets the vampire noble.
The manic pixie dream peep and the vampire noble bond over their shared love of Dear Leader.
Love develops between the manic pixie dream peep and the vampire noble.
Time passes
The vampire noble breaks up with the manic pixie dream peep over their lack of interest in immortality.
The manic pixie dream peep is distraught.
Setting: The hospital
The vampire noble's little sibling's life is threatened by cancer.
The vampire noble is distraught.
The manic pixie dream peep gives bone marrow to the vampire noble's little sibling.
The vampire noble and the vampire noble's little sibling share a tearful reunion.
The vampire noble professes their love for the manic pixie dream peep.
The manic pixie dream peep and the vampire noble live happily ever after.

Note that the current version of the game is completely gender neutral. Hence the use of terms like "manic pixie dream peep" rather than the conventional "manic pixie dream girl" and the pervasive use of the pronouns "they" and "their". This was a deliberate choice based on the fact that (1) our playtesters didn't mind roleplaying same sex relationships and/or genders other than their own, and (2) explicit gender support would require a fairly elaborate user interface for specifying player preferences for gender, heteronormativity, etc.

Related Work

There has been a considerable amount of work on story generation, within the AI and cognitive science research communities, and among authors of electronic literature (Kazemi 2013). The most relevant of these are the systems that make some attempt at knowledge representation, of which there are three main strands.

Most story generators use some kind of planning formalism. Examples include early systems such as *TALE-SPIN* (Meehan 1977), *MINSTREL* (Turner 1993; Tearse et al. 2014), and *UNIVERSE* (Lebowitz 1983). More recent systems, such as *FABULIST* (Riedl and Young 2010), have adopted more powerful planning formalisms and imported concepts from narratology, or sought to extend the planning process to encode different notions of story quality (Porteous and Cavazza 2009; Ware and Young 2011).

The planning algorithm used in *Dear Leader* is less sophisticated than that of modern story planners such as *FABULIST* or *CPOCL*. However, the story language is arguably more expressive than that found in contemporary story planners.[4] The story world generation in *Dear Leader* is also very similar to the first-phase of story generation in *UNIVERSE*, although the logic and algorithm used are different.

Another strand in story generators involves the development of explicit cognitive models, such as in *MEXICA* (Pérez y Pérez and Sharples 2001), or the design of generators influenced by models of narrative comprehension from cognitive science (Ware et al. 2014; Ware and Young 2012).

A separate, and controversial, strand in the story generation literature involves the use of story grammars (Lakoff

[3] p_{out} is required for technical reasons, but can be ignored here. It is used by other predicates calling e.

[4] For example, it's unclear how one could represent something like the `introduce_relationship` operation, which takes a relationship literal as an argument, in a language like PDDL (Kovacs 2011).

1972; Mandler and Johnson 1977; Compton, Filstrup, and Mateas 2014). The critique of story grammars is that they are not sufficiently expressive to capture the true structure of stories (Wilensky 1983). However, this depends in part on what one takes to be a "grammar." Certainly, traditional context-free grammars are insufficiently expressive, but unification grammars allow for coordination of information in distal parts of the story via communication through feature structures. And as mentioned above, the *SHOP* planner on which our system is based (Nau et al. 1999) bears distinct similarities to definite-clause grammars (Pereira and Warren 1980). Indeed, to the extent that some of the planning-based systems, such as *MINSTREL* and *UNIVERSE*, work by recombining elements from a plot library, the difference between planning- and grammar-based generators is perhaps less a sharp distinction than a spectrum of systems at different points along something like the Chomsky hierarchy.

There have been a number of recent attempts to bring story generation (broadly construed) into some kind of a mixed-initiative context. Magerko and colleagues have built systems that can collaborate with humans in traditional improv games (O'Neill et al. 2011). *Fiascomatic* (Horswill 2015) generates simple story worlds as starting points for the live-action freeform game *Fiasco* (Morningstar 2009). *The Icebound Concordance* (Reed and Garbe 2016) explores the collaborative co-generation of interactive narrative by using a story generator and exposing some of its choice points to the player. And both *Storyteller* (Benmergui 2013) and *Best Laid Plans* (Ware and Young 2015) use story planners to create gameplay based on narrative puzzles.

At a very high level, the most spiritually similar system to ours is the experimental game *Bad News* (Ryan, Samuel, and Summerville 2016). Like *Dear Leader*, it combines a procedurally generated world with human-improvisational performance. Although the story outline in *Bad News* is much more fixed than the stories generated by *Dear Leader*, the story world it generates is far more elaborate, involving an entire town of simulated characters and relationships. Samuel's notions of moving from player agency to player ownership in interactive narrative have also been an important influence in this work.

Initial playtests and future work

The replay value of Dear Leader's prototype is significantly reduced by its limited trope repertoire. Our hope is that this will change with the addition of more plot tropes and stock characters.

The main problem with the current game design is that it's easy for characters playing a beat to unwittingly introduce plot elements that are then contradicted by later beats. Since the story generation is non-interactive, there's no way for the system to redirect the story based on these elements, or even for it to be aware of them. Therefore, the primary game design challenge is to find ways of communicating to the players what *not* to do without simply giving them the entire plot in advance, which would remove the fun of not knowing what's coming. There are a number of possibilities for this. One is to have one player act as the director (i.e. a GM) to whom the entire story can be divulged in advance. Another is to try to divulge more of the story world configuration to the player in advance.

The specifics of the story world representation and the inferential power of its constraint satisfaction system are also very much in flux. While the current implementation is sufficient for the current set of tropes, characters, and settings, it will likely have to be extended in the future. In addition, some kind of well thought-out ontology for things like social relationships should be added.

Conclusion

Part of the promise of advanced game AI is to allow for more sophisticated interactive storytelling within games. However, current story generators will not be able to compete with human-authored storytelling any time soon (if this is even desirable). By building mixed-initiative story games in which player and computer share authorship of the story, the human player can compensate for whatever deficiencies the AI system might have, thereby allowing a limited story system to still produce a meaningful play experience.

Over time, successive generations of game can shift the division of labor as story generation technology improves, providing a path for incremental progress while still allowing the feedback of putting a playable experience in front of real humans.

The limitations of current interactive narrative systems make it easier in some ways to aim for comedy than drama (Horswill 2012). We have consequently aimed in *Dear Leader* for an aesthetic of irony, exaggeration, and camp. It should be acknowledged, however, that this is a risky strategy. As in games such as *Cards Against Humanity* (Dillon et al. 2011), this can encourage subversive play but can also be played for homophobia and misogyny in those who are so inclined. It is our hope that the game will not be popular with such players.

Acknowledgements

I would like to thank the Joe Bates, Spencer Florence, and Dan Feltey for being great playtesters, and the reviewers for their very helpful comments and suggestions; apologies for failing to fit them all. I'd also like to thank Ethan Robison for great discussions about tropes, and Ben Samuel for his thoughts on narrative, improvisation, and ownership.

References

Benmergui, Daniel. 2013. “Storyteller.” In *Experimental Gameplay Sessions, Game Developer’s Conference*. San Francisco, CA: UBM Techweb.

BioWare. 2012. “Mass Effect 3.”

Compton, Kate, Benjamin Filstrup, and Michael Mateas. 2014. “Tracery : Approachable Story Grammar Authoring for Casual Users.” *Papers from the 2014 AIIDE Workshop, Intelligent Narrative Technologies (7th INT, 2014)*, 64–67. http://www.aaai.org/Library/Workshops/ws14-21.php.

Dillon, Josh, Daniel Dranove, Eli Halpern, Ben Hantoot, David Munk, David Pinsof, Max Temkin, and Eliot Weinstein. 2011. “Cards Against Humanity.” Ad Magic.

Greenberg, Clement. 1939. “Avant Garde and Kitsch.” *Partisan Review* 6 (5): 34–49.

Horswill, Ian D. 2012. “Punch and Judy AI Playset: A Generative Farce Manifesto Or: The Tragical Comedy or Comical Tragedy of Predicate Calculus.” *Intelligent Narrative Techologies V (INT5)*, 14–19.

———. 2015. “Fiascomatic : A Framework for Automated Fiasco Playsets Playset Definition,” 22–28.

Kazemi, Darius. 2013. “National Novel Generation Month Repository.” https://github.com/dariusk/NaNoGenMo.

Kovacs, Daniel L. 2011. “BNF Definition of PDDL3.1.” http://www.plg.inf.uc3m.es/ipc2011-deterministic/attachments/Resources/kovacs-pddl-3.1-2011.pdf.

Lakoff, George. 1972. “The Structural Complexity of Fairy Tales.” *The Study of Man* 1: 128–90.

Lebowitz, Michael. 1983. “Creating a Story-Telling Universe.” In *Proceedings of the Eigth International Joint Conference on Artificial Intelligence*, 63–65. Karlsruhe, Germany: Morgan Kaufman.

Mandler, J.M., and N.S. Johnson. 1977. “Remembrance of Things Parsed: Story Structure and Recall.” *Cognitive Psychology* 9: 111–51.

McCoy, Joshua, Mike Treanor, Ben Samuel, Noah Wardrip-Fruin, and Michael Mateas. 2011. “Comme Il Faut: A System for Authoring Playable Social Models.” In *Proceedings of the 7th AI and Interactive Digital Entertainment*, edited by Vadim Bulitko and Mark O. Riedl. Stanford, CA: AAAI Press.

Meehan, James R. 1977. “TALE-SPIN, an Interactive Program That Writes Stories.” In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, 91–98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Montfort, Nick. 2013. *World Clock*. Cambridge, MA: Bad Quarto.

Morningstar, Jason. 2009. *Fiasco*. Durham, NC: Bully Pulpit Games.

Nau, Dana, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. 1999. “SHOP: Simple Hierarchical Ordered Planner.” In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 968–73. Stockholm, Sweden: Morgan Kaufmann Publishers Inc.

O’Neill, Brian, Andreya Piplica, Daniel Fuller, and Brian Magerko. 2011. “A Knowledge-Based Framework for the Collaborative Improvisation of Scene Introductions.” In *Proceedings of the 4th International Conference on Interactive Digital Storytelling*. Vancouver, Canada.

Pereira, Fernando C. N., and David H. D. Warren. 1980. “Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks.” *Artificial Intelligence* 13 (231-278).

Pérez y Pérez, Rafael, and Mike Sharples. 2001. “MEXICA: A Computer Model of a Cognitive Account of Creative Writing.” *Journal of Experimental and Theoretical Artificial Intelligence* 13 (2): 119–39.

Porteous, Julie, and Marc Cavazza. 2009. “Controlling Narrative Generation with Planning Trajectories: The Role of Constraints.” In *Proc. of 2nd Int. Conf. on Interactive Digital Storytelling*.

Reed, Aaron A., and Jacob Garbe. 2016. “The Ice Bound Concordance.” Santa Cruz, California: Self-published.

Riedl, Mark O, and R. Michael Young. 2010. “Narrative Planning: Balancing Plot and Character.” *Journal of Artificial Intelligence Research* 39 (217-268).

Ryan, James Owen, Ben Samuel, and Adam Summerville. 2016. “Bad News : A Game Of Death And Communication,” 160–63.

Sontag, Susan. 2013. “Notes on ‘Camp.’” In *Against Interpretation and Other Essays*. Farrar, Straus and Giroux.

Sterling, Leon, and Ehud Shapiro. 1994. *The Art of Prolog*. Cambridge, MA: MIT Press.

Swanson, Reid, and Andrew S. Gordon. 2012. “Say Anything: Using Textual Case-Based Reasoning to Enable Open-Domain Interactive Storytelling.” *ACM Transactions on Interactive Intelligent Systems* 2 (3).

Tearse, Brandon, Peter Mawhorter, Michael Mateas, and Noah Wardrip-Fruin. 2014. “Skald: Minstrel Reconstructed.” *IEEE Transactions on Computational Intelligence and AI in Games* 6: 1–10.

Turner, Scott R. 1993. “Minstrel: A Computer Model of Creativity and Storytelling.” University of California at Los Angeles.

TV Tropes community. 2004. “TV Tropes.” www.tvtropes.com.

Unity Technologies. 2004. “Unity 3D.” San Francisco, CA.

Wardrip-Fruin, Noah. 2011. *Expressive Processing: Digital Fictions, Computer Games, and Software Studies*. Cambridge, MA, USA: MIT Press.

Ware, Stephen G, and R Michael Young. 2015. “Intentionality and Conflict in The Best Laid Plans Interactive Narrative Virtual Environment” X (January): 1–11.

Ware, Stephen G., and R. Michael Young. 2011. “CPOCL: A Narrative Planner Supporting Conflict.” In *The Seventh Annual International Conference on Artificial Intelligence in Interactive Digital Entertainment*. Stanford, CA: AAAI Press.

———. 2012. “Validating a Plan-Based Model of Narrative Conflict.” In *International Conference on Foundations of Digital Games*. Raleigh, NC.

Ware, Stephen G., R. Michael Young, Brent Harrison, and David L. Roberts. 2014. “A Computational Model of Plan-Based Narrative Conflict at the Fabula Level.” *IEEE Transactions on Computational Intelligence and AI in Games* 6 (3): 271–88. doi:10.1109/TCIAIG.2013.2277051.

Wilensky, Robert. 1983. “Story Grammars versus Story Points.” *Brain and Behavioral Sciences* 6: 579–623.