Retcon: A Least-Commitment Story-World System

Ian Horswill [1]

[1] *Northwestern University, 2133 Tech Drive, Evanston, IL, 60208, USA*

Abstract
Retcon is an experimental interactive narrative engine that allows writers to author stories in which some aspects of the overall story, such as the tone, genre, and character relationships, can be chosen incrementally by the player. Stories are authored as a set of fragments (aka storylets) tagged with their story-world assumptions. The system tracks the active assumptions and removes from consideration any fragments that contradict the established story world. This allows the player some degree of co-authorship, and allows her to explore the possibility space of different stories afforded by story worlds.

Keywords
Interactive narrative, procedural generation, logic programming, constraint satisfaction, SAT

Overview

Retcon is an experimental interactive narrative system that allows players to make choices not only about plot progression, but about genre, themes, world building, and other issues of overall story design. We will refer to these issues collectively as **story state**: the changing set of design decisions during the writing process, as opposed to the state of the characters and world within the story, which we will refer to as world state.

Retcon does not allow players to design a story from scratch. It allows authors to leave elements of the story open, as a GM (game moderator) might in designing a scenario for a tabletop RPG. This gives players the ability to make limited design choices while experiencing the story, and replay to explore the implications of different choices.

The author designs the story as a set of narrative fragments, aka "storylets" [1], tagging them with the story-world elements they presuppose: character personalities, character relationships, genre, tone, theme, etc. The author then provides a set of constraints on what combinations of elements are allowable.

As with other similar systems, *retcon* starts with a story fragment, then repeatedly allowing the player to choose a valid next fragment. However, as fragments are adopted, *retcon* tracks the set of story-state assumptions they make. These, together with the constraints, define a set of possible story worlds consistent with the tale thus far. The system uses a SAT solver to determine which candidate fragments are consistent with the story so far, discarding inconsistent ones. The player then chooses from a random selection of viable fragments, the system prints the fragment's text, adds its assumptions to the story world, and repeats the process.

Example

The following are fragments from *Nana*, a narrative prototype being built using *Retcon*, in which two characters, Jaime and Joey are driving to visit Nana, the family matriarch. The character's genders are left unspecified, as is the relationship between them, which of them are related to Nana, and whether Nana is an antagonist. The constraints:

The Joint Workshop Proceedings of the 2022 Conference on Artificial Intelligence and Interactive Digital Entertainment, October 24–25, 2022, Paloma CA, USA
EMAIL: ian@northwestern.edu

```
[AtLeast 1 [Family jaime]
           [Family joey]]
[Not lovers] <- [Family jaime]
                [Family joey]]
```

stipulate that at least one of the protagonists must be a blood relation of Nana and that if they both are, they're not lovers. The rules:

```
lovers <= gay_couple
lovers <= straight_couple
```

state that, for the purposes of the story, the protagonists are lovers if and only if they're either a gay or a straight couple.

Now consider the fragment in which the protagonists remember being disciplined by Nana as children:

```
World smoking_weed
  [[Family jaime] [Family joey]].
Text smoking_weed:
  "Remember when we were kids and
   Nana caught us smoking weed?"
  "Yes, and so does my butt."
[end]
```

The `World` declaration states that the `smoking_weed` fragment presupposes that both protagonists are blood relations of Nana. Contrast this with this fragment, which assumes they are a gay couple and Nana is a homophobe:

```
World homophobe
  [[Homophobe nana] gay_couple].
Text homophobe:
  "Don't worry.  Nana's not so
   bad.  She's just a little
   'old world.'"
  "You mean she's a homophobe."
  "A <i>polite</i> homophobe."
  "I feel so much better."
[end]
```

These two fragments cannot occur in the same story world. The first assumes the protagonists are blood relations. The second states they're a gay couple; that implies they're lovers, which implies one of them is not a blood relation, although it does not imply which. Retcon can detect this kind of narrative contradiction.

In this paper, I will motivate the work by discussing the distinction between story state and world state. Then I will describe the system in technical detail. Finally, I will discuss the system's limitations: how it can and cannot be scaled.

Story State

Stories change in both the writing and the telling.

Most obviously, the story world changes as its characters move through it. They move through the space of the world and change it. They buy things, eat things, or perhaps burn down the occasional building. They acquire, achieve, and abandon goals and other involvements. They react to things, have feelings, and undergo personal transformation. This is the kind of story state that a character simulator would typically represent and reason about. It tracks the causal evolution of the story world over the story timeline as actions and other events occur within the story. For the purposes of this paper, we will call this **world state**: the state within the story world as it changes over time within the story.

But stories and story worlds can change in other, more profound ways during writing. A story intended to be a romance can become a comedy or a ghost story. The setting can change cities or time periods. Themes can be added or removed. Character backgrounds and personalities can be radically revised. Characters initially conceptualized as lovers might become platonic childhood friends. Tertiary characters can be promoted to major characters when they turn out to be more interesting than the existing characters. The primary antagonist of the television series *Twin Peaks* [2] was a stage hand accidentally shown on camera who was then written into the series.

For want of a better term, we'll call this other, more nebulous, state **story state**. One could alternatively call it "design state" or just "design" or "story". We will call it story state here to emphasize its changing nature and because it's a literal state variable in retcon.

Story state is the aspects of the design that change in real time but not in story time. The boundary between story state and world state is blurry. Moreover, they aren't independent; the story state determines the state space of the world state, the initial state of the story, and the prehistory of the story world.

In *Star Wars* [3], the protagonist Luke Skywalker moves from location to location over time, starting at his family farm on Tatooine, and moving to Yavin 4 by way of various other locations, mostly space ships. His location is world state that changes through the story. However, the fact that he grew up on Tatooine, that Tatooine is a desert planet, or even that there is a planet Tatooine is story state. It is static and unchanging within the frame of the story but evolves during the writing of the story.

It has been suggested that the story state/world state distinction is equivalent the *fabula*/*syuzhet* distinction. Although that doesn't match my understanding of the latter distinction, there may be conceptualization of *fabula* that are equivalent to story state. To the extent that *syuzhet* is a series of events within the story world, those events can conceptualized as state *transitions* within the world state, but they are neither world states themselves, nor the world state space. To the extent that *fabula* is also conceptualized as a set of events, it similarly would ben neither world state nor story state. However, to the extent that *fabula* is conceptualized as something broader, such as a set of themes or general story material, it may overlap with story state.

Serial storytelling

While conceptually distinct, world state and story state nonetheless evolve together during the writing process, with the writer regularly changing one to solve some problem with the other.

In a novel, film, or a typical narrative game, the story state evolves as the writer designs the story. Once the story is written, however, the story state becomes fixed. The audience, however, only comes to know the story state through the telling of the story. She learns the story world in piecemeal fashion as it is disclosed by individual story events. Any aspect of the story world not fixed by the story itself is left open to interpretation and interpolation by the audience, regardless of the author's intent. There is therefore play (in the sense of mobility) in the story world.

This play within the story world is leveraged in the writing of serial genres such as television, western comics, and manga. In serials, the story state continues to evolve as each new episode is written. A writer can add new characters and backstory provided her changes are consistent with the events already portrayed. These changes to the story world are treated as if they had always been true.

Improvisational storytelling

The extreme case of real-time evolution of story state is improvisational story telling. In improvisational theatre and table-top roleplaying games (TTRPGs), players are also writers. Any player can inject new material into the story state at any time. The play in the story is then not only play in the mobility sense but in the ludic sense.

Again, the primary constraint on new story state is narrative consistency: no changes can be made that violate events already play.

State in digital games

Digital narrative games, especially AI-based narratives, often maintain detailed models of world state. If there is a set of possible states the world can be in, the systems most often know the exact, specific state the world is in. Emergent narratives [4]–[6] must know the exact world state in order to forward-simulate the world. Reactive planner architectures [7], [8] similarly need the full world state to run the characters. Storylet systems [1] also assume they know the exact world state so that they can determine which story fragments are runnable at any given time.

The cost of the system knowing the exact world state is that it must commit to that world state. It has very limited latitude to change the state to suit the needs of the plot or the interests of the player.

Planning-based systems [9] need only model the specific world state elements involved in the plan. But so far as I'm aware, only Robertson's [10] system attempts to minimize state commitment. It can compensate for unexpected player behavior by retroactively changing the world state, provided those changes don't contradict anything observed by the player. It does for world state what Retcon does for story state.

By contrast, most digital narratives do relatively little reification of story state. They are like films or novels in the sense that there is a fixed story world within which one can play. One cannot change the story world as a whole beyond a certain amount of character customization and content selection.

Story state in TTRPGs

Table-top roleplaying games (TTRPGs) are situated in an opposite corner of design space. In these games, a set of human players, possibly including a special game moderator (GM) player, collaboratively improvise a story. Because any world simulation is performed by the human players, it's infeasible to model the complete world state; the cognitive load would be overwhelming.

While this makes certain kinds of gameplay infeasible, it affords greater flexibility in storytelling. It solves a number of technical problems, such as allowing a GM to simply create replacement NPCs out of whole cloth if a plot-relevant NPC is killed or incapacitated. But it also affords a great deal of collaborative freedom. If a player wants her character to be struggling with alcoholism, she can just decide that independent of the original scenario design. At the same time, if the GM has written an alcoholism subplot into the scenario and that's a trigger topic for another player, the GM can generally adapt things on the fly to remove it. Indeed, games often begin with discussions among the players about triggers, play style, narrative tone and even genre. And a good GM can improvise to adapt the story to the parameters the players agree to.

This flexibility also affords interesting game mechanics. The "aspect" system of *Fate* [11] is perhaps the best example of this. Aspects are arbitrary story world facts asserted by players; they are true by fiat. The game provides mechanics for introducing new aspects into the game world. If you're in a firefight in a warehouse, you can roll to introduce the aspect that there are oil drums in the building. If you succeed, you can set fire to them, either as a diversion, a barrier, or a weapon. The "preparedness" skill of the *GUMSHOE* system [12] is another example of a mechanic for changing the story world. It allows the players a limited ability to retroactively declare their player had planned for some contingency, optionally even including a narrative flashback describing the preparations. This saves the narrative dead time of players doing the kinds of detailed planning they would do in real life but that always happens off stage in narrative because it's tedious to listen to.

There are limits to the ability to change the story world in these games. As always, story state changes must be tone and genre appropriate and logically consistent with any story events that have already been portrayed. A player can't retroactively declare their character to be a ninja if we've already played a scene in which they were a helpless librarian, short of major complications in the story world such as the multiverse of *Everything, Everywhere, All At Once* [13]. Introducing an AR-15 rifle into a *Dungeons and Dragons*™ game, or an elf into a science fiction game would generally not be considered genre appropriate and would at least require buy-in from the table.

These kinds of changes to story state are practical for TTRPGs because the simulation is being performed by human players with human-level intelligence. They can easily decide what kinds of changes would or would not contradict the narrative.

Simulating this kind of gameplay in a digital game is well beyond current AI capability. However, there is a clear opportunity to expand the aesthetic possibilities of digital narrative by implementing limited subsets of this kind of narrative play. *Retcon* is one attempt to do this.

Retcon

Retcon is a first attempt to duplicate a few of the affordances of TTRPGs in a digital narrative. It does not allow players to change the historical world state of the game, as does Robertson's system. Nor is it generative enough to allow players to create new facts to inject in the story world. However, it does allow the author to leave open specific story choices about genre, tone, style, and the nature of the characters. Players can then determine these choices incrementally as the story progresses.

Retcon is a storylet system [1], meaning the author writes the story in terms of a set of narrative fragments. In *retcon*, the author also tags fragments with the story-world assumptions the fragments presuppose. The author also provides constraints on what assumptions are compatible with one another. If a fragment introduces a werewolf, then we know we're in a horror story, or at least a supernatural story. If some other fragment has established that we are in hard science fiction, then the werewolf fragment is an invalid continuation for that story.

As the story progresses, the system tracks the assumptions that have been established and removes incompatible fragments from consideration.

Formalization

We will represent story states as sets of assumptions about the story world in some logic. Each successive beat of the story introduces new assumptions about the story world, thus changing the story state, and narrowing the space of possible story worlds. A *retcon* story specifies:

- A logic for describing story state. We will use the symbol L to denote the logic's "language," i.e. the set possible statements in the logic.[2]
- A theory, $T \subset L$, describing the restrictions on possible story worlds.[3]
- A set of story fragments, F
- An assumption function, $A: F \to 2^L$, assigning to each fragment f, a set of story-world assumptions, $A(f)$.

We will give the details of the specific logic used in *retcon* and the types of statements supported for the theory and assumptions in the implementation section, below. The remainder of this section can be thought of as formalizing what it means for something to be "*retcon*-like" before getting into the specific version that's currently implemented.

Possible story worlds

Since T is essentially a set of constraints on valid story worlds, the class of possible story worlds is the set of models of T. For any set of statements $S \subset L$, let $\mathcal{M}(S)$ denote its set of possible models $\mathcal{M}(S) = \{ M | M \vDash S \}$. The set of all possible story worlds is then $\mathcal{M}(T)$. Since each successive fragment adds assumptions, it narrows the possible story worlds.

Story states

In *retcon*, the story state at some point in the writing/telling of a story is the set of story-world assumptions in play at that moment. These determine the set of possible story worlds at that moment. If some set of fragments $f_1, \dots, f_n$ have been used in the story up to that point, then the story state is the union of assumptions due to those fragments: $A(f_1) \cup \dots \cup A(f_n)$. And the remaining story states are then:[4]

$$\mathcal{M}\big(T \cup A(f_1) \cup \dots \cup A(f_n)\big)$$
$$= \mathcal{M}(T) \cap \mathcal{M}\big(A(f_1)\big) \cap \dots \cap \mathcal{M}(A(f_n))$$

Crucially, we will never have to compute $\mathcal{M}$ explicitly; we need only determine whether it is empty for a given set of assumptions, and this can be done using a SAT solver. If the solver fails, then there are no possible story worlds, and the assumptions are mutually contradictory.

Through abuse of notation, we will adopt use the $\Diamond$ operator of modal logic to indicate satisfiability/possibility. $\Diamond P$ means "P is possible":

$$\Diamond P \Leftrightarrow \mathcal{M}(P) \neq \emptyset$$

Algorithmically, $\Diamond P$ simply means "the SAT solver can find a model for P."

World states

Some formalization of mutable world state is generally necessary to prevent nonsensical beat sequences. While such state is necessary, we do not claim to have a better representation of mutable state. We simply assume that there is:

- A set of possible world states, W. These might be the models of some logical theory or not; we don't assume any particular structure for world state.
- A precondition function, $P: F \to 2^W$, indicating in which world states a given fragment is valid
- A transition function, $\delta: F \times W \to W$ indicating how a given fragment updates the world state.

Fragment validity

Again, the story consists primarily of a stock of fragments. However, not all fragments will make sense at a given point in the story. Fragments can be ruled out either because their world-state preconditions are invalid, or because

[2] The language of a logic is the set of grammatical strings in the logic, also known as its "well-formed formulae" or "WFFs." Saying something is a WFF doesn't say anything about it being true or false.
[3] A theory in logic parlance is just a set of statements in the logic so $T \subset L$.

[4] Note the equality here is not valid for stable-model semantics, so this formalization would have to be modified to use answer-set programming.

their assumptions are inconsistent with the current story state (which is itself a set of assumptions). A fragment $f \in F$ is valid in story state $s \subset 2^L$ and world state $w \in W$, if and only if its preconditions are satisfied and there is at least one possible story world in which the current assumptions and the fragment's assumptions are both valid:

$$P(f, w) \wedge \ \Diamond(T \wedge s \wedge A(f))$$

Basic algorithm

Retcon begins with an initial world state, $i \in W$, and an empty story state. It then iteratively chooses a set of valid fragments, prompts the user to choose one, and updates the world and story state accordingly:

```
$w = i$
$s = \emptyset$
repeat until end of story
  $C = \{ f \mid P(f, w) \wedge \ \Diamond(T \wedge s \wedge A(f))\}$
  Present some subset of $C$ to the player
  Player chooses some specific fragment $f$
  Print text of $f$
  $w = \delta(f, w)$
  $s = s \cup A(f)$
end
```

The test for $\mathcal{M}$ being non-empty is equivalent to a test of whether $T \cup s \cup A(f)$ is satisfiable/consistent. It can therefore be tested using a satisfiability solver.

Implementation

Retcon is implemented as an embedded language in the *Step* programming language [14]. Stories are defined through a mixture of normal *Step* code and special language features added by retcon. The implementation presently runs in the *Unity* game engine [15]. Satisfiability is tested using *CatSAT* [16], which is a randomized SAT/SMT solver that can run natively in *Unity*. *CatSAT* isn't an ideal choice for retcon, but it works well enough for our purposes.

World state

World state is implemented using the *Step*'s state-tracking features, which support both conventional mutable variables and fluent predicates that can be updated imperatively. Updates are rolled back upon backtracking. For more information, see [14].

Story state logic

Since we are using a SAT solver to test satisfiability of world states, story states and world theories must be reducible to finite Boolean expressions. This means the underlying logic cannot be full first-order logic. We limit the logic to pseudo-Boolean constraints universally quantified over some finite domain. This means it is similar to ASP, but all models of classical logic are allowed, not merely the so-called "stable" models. In particular, the author defines:

- A specific set of possible story objects, U, to reason over (characters, items, locations, *etc.*)
- Specific types or collections of those objects. These are used to quantify variables over those types
- A specific set of predicates over those objects

The logic has the following structure:

- Terms are then either constants (elements of U) such as `nana`, `joey`, `the_farm`, or variables: `?x`, `?y`, `?character`. We do not support term expressions.
- Atoms are therefore predicates applied to these terms: `[Loves nana, joey]`, `[At jaime ?location]`, etc. Atoms containing no variables are called ground atoms.
- Literals are atoms or negated atoms: `[Loves ?x nana]`, `[Not [Loves ?x nana]]`. Similarly, ground literals are literals containing no variables.
- If $L_1, \ldots, L_n$ are literals, then `[Unique` $L_1 \ldots L_n$`]` states that exactly one of the literals must be true. `[AtMost` n $L_1 \ldots L_n$`]` states that at most n may be true. Other versions, such as `AtLeast`, `Exactly`, etc. are also supported
- If C and $A_1, \ldots, A_n$ are literals, then:
- $C \leftarrow A_1 \ldots A_n$ means $A_1 \wedge \ldots \wedge A_n$ imply C
- $C \Leftarrow A_1 \ldots A_n$ means $A_1 \wedge \ldots \wedge A_n$ imply C but also that C implies at least one righthand side of a $C \Leftarrow$ expression.

- Prefixing any of the above statements with a collection applied to a variable universally quantifies that variable over the collection. Thus:
  `[Character ?x] [Loves ?x nana]`
  means “everyone loves Nana.”

With those preliminaries, we can now state the expressive capabilities of the current implemented system:

- Story world constraints can be any of the expressions above, provided all variables are universally quantified over some collection.
- Fragment assumptions can be any (possibly empty) set of ground literals and/or fully quantified literals. However, they cannot be cardinality constraints or implications.

Related Work

Although I am not aware of prior work on dynamic story state, there have been several papers that have used algorithms with similar motivations to solve adjacent problems.

GME [10] reasons about multiple world states in order to facilitate recovery from problematic player actions in a planning-based story framework. The *ISR* narrative planner [17] can change its initial world state to improve story generation.

AutoDread [18] uses a SAT system to guide the player through a character design questionnaire, removing potential answers that contradict previously established facts or removing questions entirely if their premises contradict or are implied by established facts.

RoleModel [19] is a story generator that allows the roles of characters within the story to be specified at runtime, then uses answer set programming [20] to choose instantiations of story skeletons with those roles.

Benmergui’s *Storyteller* [21] is a narrative puzzle game in which players are challenged to complete a partially specified narrative to achieve a specified narrative goal. The system uses constraint satisfaction to dynamically adjust parts of the narrative.

Several interactive narrative systems have looked to logic programming to support sophisticated character reasoning about world state and social interactions. *Versu* [8] used a bespoke logic to allow character not only to choose actions but to render judgements about the actions of other characters. The commercial game *City of Gangsters* [22], [23] uses a more conventional logic programming language but for a large-scale simulation involving over a thousand concurrent NPCs. *Ceptre* [24] uses linear logic to reason about world state, but could also perhaps be used to reason about dynamic story state.

Many interactive narrative systems work by assembling stories from fragments [25]–[32]. These have been referred to as “storylet” or “content selection” architectures. Kreminski and Wardrip-Fruin offer a recent survey [1]. They share some mechanism for defining fragments that can generate text, a method for assigning preconditions to them, and a method for choosing a next fragment. The basic framework used in *retcon* could be added to any of these systems; it simply adds a new precondition mechanism to the fragments. Alternatively, these architectures could be implemented within *retcon*; it is agnostic as to content selection and precondition architecture.

The most common preconditions are so-called story “qualities” [27]. These are state variables, typically numeric, which can be set by fragments and tested in preconditions. However other, more elaborate, systems have been implemented, including general logical queries against a knowledge-base [25].

Content-selection algorithms range from random [30] to A* pathfinding [33] and full reactive planning [25].

Future work

Retcon uses a relatively simple logic. Some surface restrictions can be ameliorated by normalizing a superficially more expressive logic into the existing logic. For example, the current system only allows ground literals to be used as fragment assumptions. However, a more complex fragment assumption Φ can be supported simply by substituting a proposition p for it, and adding the statement $p \leftrightarrow \Phi$ to T. Thus far, our use cases have not required this.

CatSAT is not an ideal choice of a SAT solver for this application. It is optimized for generation of random models within a small memory footprint. Since *retcon* doesn’t even look at the model that is generated, a CDCL SAT solver such as *Z3* [34] would be preferable. But again, *CatSAT* has been sufficient for our use cases.

Another possibility would be to use an answer-set solver, such as *Clingo* [35]. This has the advantage of providing a high-performance solver and a stronger logic, at the cost of more complicated integration with the rest of the game, and the need to generate the AnsProlog code from whatever format the story is authored in. It would also require redesigning the formalization given in this paper, which assumes a monotonic logic.

Conclusion

Interactive narrative systems have focused primarily on reasoning about world state. *Retcon* demonstrates that it is possible to build story systems that dynamically manipulate the core assumptions of the story world, allowing the player some agency in choosing them.

The boundary between story state and world state is somewhat artificial. Any part of the world state that doesn't change during the story can be considered story state. Conversely most story state could be incorporated as an aspect of world state that simply doesn't change during the story. In principle, this kind of constraint reasoning could also be done over world state. However, general constraint-reasoning over state *histories* would be extremely expensive. Constraint reasoning over story state is much more practical because it is quasi-static.

References

[1] M. Kreminski and N. Wardrip-Fruin, "Sketching a Map of the Storylets Design Space," in *International Conference on Interactive Digital Storytelling (ICIDS-18)*, 2018, pp. 160–164, doi: 10.1007/978-3-030-04028-4.

[2] M. Frost and D. Lynch, "Twin Peaks," CBS Television Distribution, USA, 1990.

[3] G. Lucas, *Star Wars*. USA: 20th Century Fox, 1977.

[4] J. Ryan, "Curating Simulated Storyworlds," University of California Santa Crus, 2018.

[5] Maxis, "The Sims 3." 2009.

[6] T. Adams and Z. Adams, "Slaves to Armok: God of Blood Chapter II: Dwarf Fortress." Bay 12 Games, 2006.

[7] M. Mateas and A. Stern, "A Behavior Language for Story-Based Believable Agents," *IEEE Intell. Syst.*, 2002, doi: 10.1109/MIS.2002.1024751.

[8] R. Evans and E. Short, "Versu - A Simulationist Storytelling System," *IEEE Trans. Comput. Intell. AI Games*, vol. 6, no. 2, pp. 113–130, 2014.

[9] S. G. Ware and R. M. Young, "CPOCL: A Narrative Planner Supporting Conflict," 2011.

[10] J. Robertson and R. M. Young, "Interactive Narrative Intervention Alibis through Domain Revision," *AIIDE*, 2015.

[11] L. Balsera, B. Engard, J. Keller, R. Macklyn, and M. Olsen, *Fate Core*. Evil Hat Productions, 2013.

[12] R. Laws, "GUMSHOE System Reference Document," London, UK, 2013. [Online]. Available: http://site.pelgranepress.com/index.php/the-gumshoe-system-reference-document/.

[13] D. Kwan and D. Scheinert, *Everything Everywhere All At Once*. A24, 2022.

[14] I. Horswill, "Step: A Highly Expressive Text Generation Language," 2022.

[15] Unity Technologies, "Unity 3D." San Francisco, CA, 2004.

[16] I. Horswill, "CatSAT: A Practical, Embedded, SAT Language for Runtime PCG," 2018.

[17] M. Riedl and R. M. Young, "Open-World Planning for Story Generation," 2005.

[18] I. Horswill and E. Robison, "What's the Worst Thing You've Ever Done at a Conference? Operationalizing Dread's Questionnaire Mechanic," in *AIIDE-18 Workshop on Experimental AI in Games (EXAG-18)*, 2018, no. Horswill.

[19] S. Chen, A. M. Smith, A. Jhala, N. Wardrip-Fruin, and M. Mateas, "RoleModel: Towards a Formal Model of Dramatic Roles for Story Generation," 2010.

[20] V. Lifschitz, *Answer Set Programming*, 1st ed. Springer, 2019.

[21] D. Benmergui, "Storyteller," 2013.

[22] R. Zubek and M. Viglione, "City of Gangsters." Kasedo Games, 2021.

[23] R. Zubek, I. Horswill, E. Robison, and M. Viglione, "Social Modeling via Logic Programming in City of Gangsters," 2021.

[24] C. Martens, "Programming Interactive Worlds with Linear Logic," Carnegie Mellon University, 2015.

[25] M. Mateas and A. Stern, "Façade." 2005.

[26] A. Kennedy, "Fallen London." Failbetter Games, London, 2009.

[27] A. Kennedy, "StoryNexus." Failbetter

Games, London, 2009.

[28] D. Sharp, “The King of Chicago.” Cinemaware, 1987.

[29] A. A. Reed and J. Garbe, “The Ice Bound Concordance.” Self-published, Santa Cruz, California, 2016.

[30] F. Alliot, “Reigns.” Devolver Digital, Austin, TX, 2016.

[31] J. Garbe, M. Kreminski, B. Samuel, N. Wardrip-fruin, and M. Mateas, “StoryAssembler : An Engine for Generating Dynamic Choice-Driven Narratives,” in *Foundations of Digital Games (FDG)*, 2019, p. August.

[32] R. Colantonio, “Weird West.” Devolver Digital, Austin, TX, 2022.

[33] E. Short, “NPC Dialog Systems,” *IF Theory Reader*. > Transcript On Press, Boston, MA, pp. 331–358, 2011.

[34] L. De Moura and N. Bjørner, “Z3: An efficient SMT Solver,” 2008, doi: 10.1007/978-3-540-78800-3_24.

[35] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele, “A User ’ s Guide to gringo , clasp , clingo , and iclingo ∗,” Potsdam, 2010.