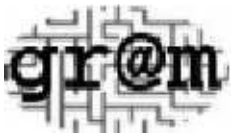
McGill University
School of Computer Science
Game Research at McGill



Analyzing Modern Computer Game Narratives

GR@M Technical Report No. *2010-1*

Clark Verbrugge Peng Zhang
School of Computer Science, McGill University
Montréal, Québec, Canada, H3A 2A7
clump@cs.mcgill.ca pzhang2@cs.mcgill.ca

March, 2010

Abstract

In many computer games, narrative is a core component, with the game centering on an unfolding, interactive storyline which both motivates and is driven by the game-play. For larger games, however, narrative size and complexity makes analysis difficult, and while formal approaches to assessing game narratives and identifying flaws have been proposed, scalability remains a barrier to practical use. In this work we develop an optimized, formal analysis system for interactive fiction narratives. Our approach is based on a relatively high-level game language, borrowing analysis techniques from compiler optimization to improve performance. We demonstrate our system on a variety of non-trivial narratives analyzing a basic reachability problem, the path to win the game. The results are very encouraging, and using our optimized design we are able to analyze narratives orders of magnitude larger than the previous state-of-the-art. This level of performance allows for verification of narrative properties at practical scales.

1 Introduction

Narrative is a central feature of many computer games, extending from text-based interactive fiction to current adventure, RPG, and mixed genres. For such games a number of narrative properties become important to an immersive game experience, including logical consistency, continuity of story elements [22], level of "tension" or atmosphere [6], as well as game-play issues such as ensuring player progress, and adequate coverage of potential player choices [1, 29].

The complexity of larger narratives, however, can make formal analysis of narratives difficult. Some success has been had in adaptively preserving game properties during development or interactive generation [26, 22], but for analysis of longer game segments or existing, complete narratives the combinatorial explosion implied by a rich set of player choices and game content elements quickly results in scaling issues, limiting conservative verification of narrative properties.

In this work we develop an efficient system for analyzing complex game narratives. Our approach is based on an exhaustive analysis of the narrative state-space, and thus applies to general reachability problems. To improve scalability, we extract high-level game information from game code by applying program analysis techniques more traditionally found in the compiler optimization domain. This information is used within an optimized search to reduce the branching factor and improve performance.

We illustrate our design by examining a fundamental example of reachability, the path to win the game. In this context we analyze a variety of non-trivial narratives derived from both commercial and amateur interactive fiction games. Using our optimized state search, we are able to determine winnability within seconds to minutes on a modern machine. These results dramatically improve on previous, low-level work on formal winnability analysis, which has been limited to narrative inputs orders of magnitude smaller [23]. The ability to perform reachability analyses on narratives of much larger scales shows that formal verificaton of a wide variety of properties can be feasibly performed on non-trivial, industrial-size game narratives.

1.1 Contributions

Specific contributions of this work include:

- We present a complete, automatic analysis system for computing reachability on computer game narratives.
- Our design is optimized by a novel application of compiler dataflow analysis techniques, developing several new analyses specific to game narratives.
- We provide detailed experimental results on a variety of non-trivial narratives considering the basic problem of winnability. As well as demonstrating analysis performance these extensive results reveal interesting behaviours useful for further understanding game narratives.

In the next section we give details on the narrative model we use in our study. Section 3 describes our optimized analysis system, and Section 4 gives experimental results. Related work is described in Section 5, followed by conclusions and future work.

2 Narrative Model

Analysis of game narratives assumes of course a suitable representation. We make use of a language for constructing interactive fiction (IF) games, *PNFG* [23]. The IF genre has the advantage of consisting of complex narratives, relatively easily extracted from the game architecture. The PNFG format is preferred over more full-featured languages such as Inform [19] as a simplified, but complete IF language with a well-defined semantics and reduced syntax.

Below we describe the overall design and common properties of interactive fiction games, followed by details on the PNFG language that we exploit in later analysis.

2.1 IF Properties

An interactive fiction narrative provides a minimal, typically text-based virtual environment. A player avatar is controlled through textual input forming game commands, and the current or resulting game state is reported by textual output. Game-play is turn-based, and usually involves exploration of an interconnected series of rooms, wherein the player (avatar) may examine, move, and otherwise manipulate game objects. Appropriate actions unlock or control narrative progress, moving the player from an initial state to either a winning or losing conclusion.

Figure 1 gives a basic overview of game control-flow. After initialization, the game waits in an idle state for user input; user commands trigger game actions, which can result in either a game win, loss, or (more typically) a return to the idle state following any post-turn processing.

The complexity of game state is an important property to analysis. In IF games critical game state consists of simple object properties, such as a room being lit or unlit, as well as the object containment hierarchy—the location of each object, including the player's inventory. Counters and dynamic object allocation can add further complexity, although games are usually finitely bounded, with a small limits on counters and a fixed maximum number of available game objects.

2.2 PNFG

The PNFG language provides a minimal model of IF game structure, directly implementing the control-flow model shown in Figure 1. Game objects and rooms are defined along with boolean

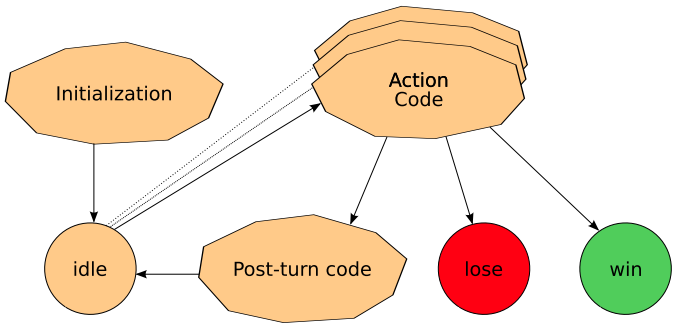


Figure 1: *IF control flow.*

state variables, and containment is internally represented by further boolean state (x (not) in y for all objects x and locations y). User commands invoke code which can set and unset object states, move objects between locations, as well as branch conditionally on object state or location. Syntatic sugar is provided for a number of further common IF constructs, including counters, scoping of game commands; more language details can be found in [23].

PNFG code maps directly to a structured Petri net. This permits low-level analysis, but also ensures a simple syntax, readily amenable to high-level analysis approaches. Higher-level game knowledge enables a number of optimizations that significantly improve the ability to search the state space of the game narrative, and thus the ability to validate game properties. In the next section we describe our analysis system and associated optimizations based on exploiting knowledge gained from narrative source code and semantics.

3 Narrative Analysis

Our overall approach to narrative analysis is illustrated in Figure 2. PNFG game specifications are subject to initial, high-level analysis, which is then used in conjunction with the game's interpretable (compiled "NFG") form as part of an optimized search of the game state space. The subsections below describe the basic search behaviour, as well each of our advanced optimizations.

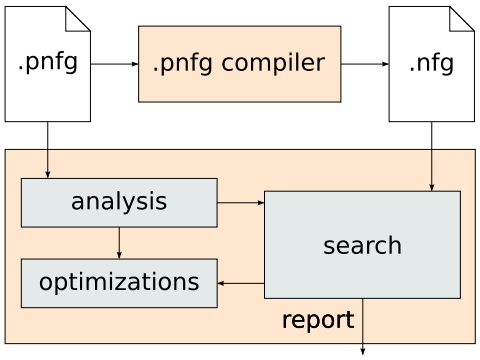


Figure 2: *Overall system design.*

As an example of useful narrative analysis, and to give our study a specific analysis goal, we investigate the general problem of computing "winnability." This involves identifying the sequence

of commands that bring a player from an initial state to a winning game completion. As well as verifying a fundamental game property (the game should be winnable), finding one or all "winning paths" is an instance of the more general problem of efficiently determining state reachability, and our techniques can be easily abstracted from the winnability goal and applied to other search problems.

3.1 Basic search

The basic state-space search proceeds as a back-tracking, depth-first search of reachable game states, applying all possible game commands at each state. Game state is internally stored in terms of the state of the internal NFG representation, but is not directly accessed by the search system. Reachability and progress are instead determined by the ability of a game interpreter incorporated into the system to apply a given command, monitoring for win, lose or error conditions.

A number of generic optimizations improve performance of this basic, brute-force approach. Cycle detection is an effective tool for most narratives. Many games, for example, tend to permit a variety of commands such as "look," "examine," or command sequences such as "take x, drop x" that do not overall modify game state, but nevertheless contribute to the branching factor and lengthen search paths. Our system thus maintains a stack of states during its DFS activity, and truncates searches containing states previously encountered in the current game search path. Caching is also used across search branches in order to avoid "dead-ends," or states which have no legal actions that can lead to a game win (i.e., all actions lead to either losing, errors, or dead-ends). These states are cached, and used to further prune the state space search. Since PNFG includes a simple static scoping model for game actions, the branching factor is further reduced by limiting the commands tried at each state to those available in the current game scope.

Although these optimizations are effective, larger narratives have many commands and objects as well as potentially long solution depths, and for practical analysis a very small branching factor and/or good search heuristics are required. Our approach applies dataflow analysis techniques, more typically used in compiler optimization, to extract game information in order to further improve performance. Below we describe our main analysis stages, beginning from our most core optimization.

3.2 Pre/Post-condition Analysis

The basic search process prunes illegal commands, to some extent by tracking static program execution scope, but primarily by dynamically identifying game error states. Legal combinations of commands, however, can also be recognized prior to the actual search by observing the requirements built into the state modifications of each game action. This reduces the cost of encountering bad command pairs and pressure on the dead-end cache.

In the PNFG language, many of the operators that modify game state, either by changing boolean variables or by moving an object from one location to another, assume a specific input state. For example, the command to change a game object variable from false to true (`+object.var`) requires the variable be initially false, and similarly, the object move statement (`move x from y to z`) assumes the object `x` is currently in `y`. Note that successful execution also guarantees the resulting output state: in the first example `object.var` is certainly true if the statement completes without error, and in the second `x` will be in `z`.

```
(you,take,candle) {
                                                    candle-in-desk:⊤, candle-in-you:⊤, candle.lit:⊤, you.hurt:⊤
  move candle from desk to you;
                                                  candle-in-desk:false, candle-in-you:true, candle.lit:⊤, you.hurt:⊤
  if (!candle.lit) {
                                                candle-in-desk:false, candle-in-you:true, candle.lit:false, you.hurt:⊤
   +candle.lit;
                                                 candle-in-desk:false, candle-in-you:true, candle.lit:true, you.hurt:⊤
  } else {
                                                 candle-in-desk:false, candle-in-you:true, candle.lit:true, you.hurt:⊤
   +you.hurt;
                                               candle-in-desk:false, candle-in-you:true, candle.lit:true, you.hurt:true
  }
                                                 candle-in-desk:false, candle-in-you:true, candle.lit:true, you.hurt:⊤
 }
```

Figure 3: An example of post-condition analysis: dataflow information (on the right) is propagated in a forward direction.

These observations drive two basic analyses which we then combine in order to prune the branching factor. We first analyze each action in a *backward* direction, computing a conservative approximation of the minimal necessary conditions for the action to execute correctly. We then perform a very similar analysis in a *forward* direction, conservatively computing the output conditions. In order for one action to follow another then, output of the first must be compatible with the input requirements of the second.

In a formal sense these form symmetric dataflow problems. If we consider our forward, *post-condition* analysis, we associate with each object variable and x, y object location state (x in y) a value from the domain {true,false,$\top$}; *i.e.,* the (incomparable) elements true and false, along with a greatest element $\top$. The latter represents a state which may be either true or false. Note that as a simplification over traditional dataflow we do not need a smallest element, $\bot$, since we treat lack of information ($\bot$) and inconsistent information ($\top$) as identical in our action matching process.

The dataflow technique propagates these pairings through all code paths starting from the action code entry (or exit) with all variables in unknown/inconsistent states ($\top$), modifying the associated domain values according to the game action. The statement `+object.var`, for example, will in a forward sense result in an output pairing `object.var:true`, and in a backward sense the pairing `object.var:false`. At conditionals information is duplicated along each branch, and merged by setting `object.state:`$\top$ whenever `object.var` is not the same on both sides.

A short example of the application of pre and post-condition analysis is shown in Figures 3 and 4. In this case post-condition analysis is reasonably effective, determining the location and the state of the candle, although not the state of the player. Pre-condition analysis is limited by merging in a reverse direction, at the conditional test itself where the branching conditions are discovered too late to improve information; here it is only able to determine that the candle must be in the desk.

Upon completion of this analysis, each action has calculated pre and post-conditions. A variable with a post-condition of true is necessarily true on action exit, and a variable with a pre-condition of true must be true on input if the action is to execute correctly; symmetric for false of course. For each action pair compatibility can thus be easily tested: if the post-conditions of action A include

```
(you,take,candle) {                                    candle-in-desk:true, candle-in-you:false, candle.lit:⊤, you.hurt:⊤
  move candle from desk to you;                        candle-in-desk:⊤, candle-in-you:⊤, candle.lit:⊤, you.hurt:⊤
  if (!candle.lit) {                                   candle-in-desk:⊤, candle-in-you:⊤, candle.lit:false, you.hurt:⊤
   +candle.lit;                                        candle-in-desk:⊤, candle-in-you:⊤, candle.lit:⊤, you.hurt:⊤
  } else {                                             candle-in-desk:⊤, candle-in-you:⊤, candle.lit:⊤, you.hurt:false
   +you.hurt;                                          candle-in-desk:⊤, candle-in-you:⊤, candle.lit:⊤, you.hurt:⊤
  }                                                    candle-in-desk:⊤, candle-in-you:⊤, candle.lit:⊤, you.hurt:⊤
 }
```

Figure 4: An example of pre-condition analysis: dataflow information (on the right) is propagated in a backward direction.

`object.var:`a and pre-conditions of B have `object.var:`b for a given object variable, then action B can follow action A as long as $a \sqsubseteq b$ or $b \sqsubseteq a$. Actions available at each point in the state search are thus selected according to these restrictions. In our example from Figures 3 and 4, the action is not compatible with itself, since we have both `candle-in-you:true` and `candle-in-desk:false` in post-conditions, while pre-conditions require `candle-in-you:false` and `candle-in-desk:true`.

3.3 Winning Set

In most games the majority of game components and interactions are not in fact essential to winning the game. Some number of unnecessary objects and commands are usually provided in order to ensure solution complexity and to provide a greater sense of immersion. The branching factor in a search for win may thus be further reduced by eliminating commands that would not appear in any optimal winning solution.

The "winning set" of commands is conservatively identified by a basic source code analysis, starting from a statement provided by PNFG to win the game: `+game.win`. We maintain two sets, one of winning actions, and one of goal statements. The latter consists initially of just the (`+game.win`) statement, and the former includes all actions that contain a `+game.win` statement.

The winning set is then calculated in a transitive sense based on the code structure of the statements that cause an action to be added. PNFG action code does not contain loops (other than as syntactic sugar for duplication), and so we associate each game statement with a (finite) list of the conditional tests in which it is nested. For example, in the PNFG code

```
if (x.y) {
  if (a contains b) {
    +game.win;
  }
}
```

the win statement is nested within two conditional tests. Thus, in order to reach the win statement, the player must also at some point have executed an action which sets `x.y` to true, and one which moves `b` into `a`. These two operations are added to the goal statements, and all actions containing them (semantically) are added to the winning set. These actions and goal statements are then recursively processed, eventually closing in on the set of commands (and game states) that may be necessary to win the game, and excluding all commands definitely not required.

3.4 Uselessness

A similar observation drives a separate "uselessness" analysis. The existence of superfluous game content means many game objects are not required in order to win the game. The most trivial instances of uselessness, where an object is never referenced at all by code in the winning set will be discovered as part of a winning set calculation. Many further useless objects, however, may be referenced or otherwise acted upon in winning actions, even if they are not required to win the game. Here we address uselessness as a distinct problem from the winning set.

We base our "uselessness" analysis on several recursive definitions. The uselessness of an object depends on the uselessness of both its location component, as well as all associated object variables, and both also depend on the data not being used in useful action contexts. We thus define the following:

1. A *useless variable* of an object is a variable which is read only by useless actions, or is only read in order to assign to itself.
2. A *useless object* is an object for which the location is only referenced by useless actions.
3. A *useless action* is an action which modifies only useless objects and variables.

The extra condition on useless variables accommodates uses of variables as flags guarding textual changes, such as encountered in the common idiom of showing the player a complete description of a room or item only on first encounter, and an abbreviated description subsequently.

From these definitions we compute a greatest fixed-point. We initially consider all objects, variables and actions useless, except for game critical objects such as the player, the root location, and the game win/lose variables. This set is refined by iteratively discovering and removing non-useless elements. The resulting set is used to prune the actions and states considered in the search.

3.5 Accurate Match

In PNFG code, and IF games in general, actions often have constraints for execution, encoded as conditional tests that guard the entire action body. This makes the action code conditional, however, and due to conservative merging, from the perspective of our pre-condition analysis such actions will have no input constraints, and may thus follow any other action. Such "conditional actions" add to the branching factor and just result in excess backtracking during search, as the conditional move is repeatedly tried (and rejected as resulting in a cyclic state) in invalid contexts.

Our "accurate match" heuristic attempts to identify and improve handling of conditional actions. Actions outwardly consisting of a single conditional test, with no "else" branch are separately

Benchmark	LoC	Rooms	Objs	Vars	Global Cmds	Room Cmds	Depth
ageorg15	1230	9	16	33	15	45	17
dprykh	1472	8	11	23	40	29	13
hsafad	387	10	14	18	2	39	12
mcheva	775	14	10	6	0	62	26
sdesja8	775	10	11	35	0	51	17
RTZ Chap 1	583	11	20	10	5	42	11
RTZ Chap 2	1113	22	37	16	4	118	19

Table 1: Benchmarks and properties: lines of code, total number of rooms, objects, boolean object variables, commands (global and room-local), and minimum solution depth.

identified and their conditions added to the pre-condition constraints. Although this is a simple pattern-based approach, it encompasses a large number of cases.

4 Experimental Analysis

We have implemented our system and examined the effects of our optimizations on a variety of moderate size narratives, roughly the size of a commercial game chapter. In each case we measure the performance of the analysis system as it is applied to a basic winning path problem. Below we describe our benchmark suite, discuss our measurement strategy, and present experimental results and accompanying observations.

4.1 Benchmarks

Benchmarks are drawn from two main sources: the well-known commercial game, *Return to Zork* [4], where we have modeled the first two chapters in PNFG, and several new narratives developed directly in PNFG by undergraduate and graduate students as part of a course assignment. The latter represent amateur efforts, but were required to respect some basic complexity measures, including lower bounds on number of objects, commands, solution complexity and length. Table 1 summarizes interesting, static properties of our benchmarks. Naively, the state graph to be searched will have a branching factor given by the number of commands (global and room-specific), and a depth at least of the solution depth. Assuming any object can be in any room and any room can be in any other room, the overall state space is bounded in size by $|\text{Rooms}|^{|\text{Objs}|} \times (|\text{Rooms}| - 1)^{|\text{Rooms}|} \times 2^{|\text{Vars}|}$.

4.2 Measurements

We measure performance of winnability analysis on our benchmarks under our basic system, as well as with various individual and combinations of optimizations. Measurement of winnability analysis is complicated by several factors. An exhaustive test, reporting every winning path up to a given depth would, for instance, provide an equal workload to each optimization. Complete enumeration of winning solutions, however, is not practical—the number of possible solutions grows very quickly

as search depth increases, resulting in performance being quickly dominated more by reporting time than analysis parameters. We avoid this problem at the cost of greater variance by measuring only the time to find the *first* winning solution. In order to avoid skew introduced by the order in which nodes are examined in the search, we also randomize the order of actions considered as each node is expanded in the search.

Results are shown in Figures 5 through 11. For each benchmark and optimization combination, and for a range of maximum search depths, a series of 10 analysis attempts are performed and the average time plotted. Note that while general trends are stable variance can be significant, as a fortunate or unfortunate node ordering during search can have a large impact on an individual experiment. All results were gathered on the same quad-core Xeon 2.3GHz machine with 16GB RAM, Debian 2.6.18, using Java HotSpot 1.5.0_14 and a 1.5GB heap, and were limited to 5 minutes per search (except for DPRYKH at 6 minutes). Also note that we do not report results without our basic pre/post-condition analysis since with the exception of RTZ CHAPTER 1 none complete within our 5 minute timeout.

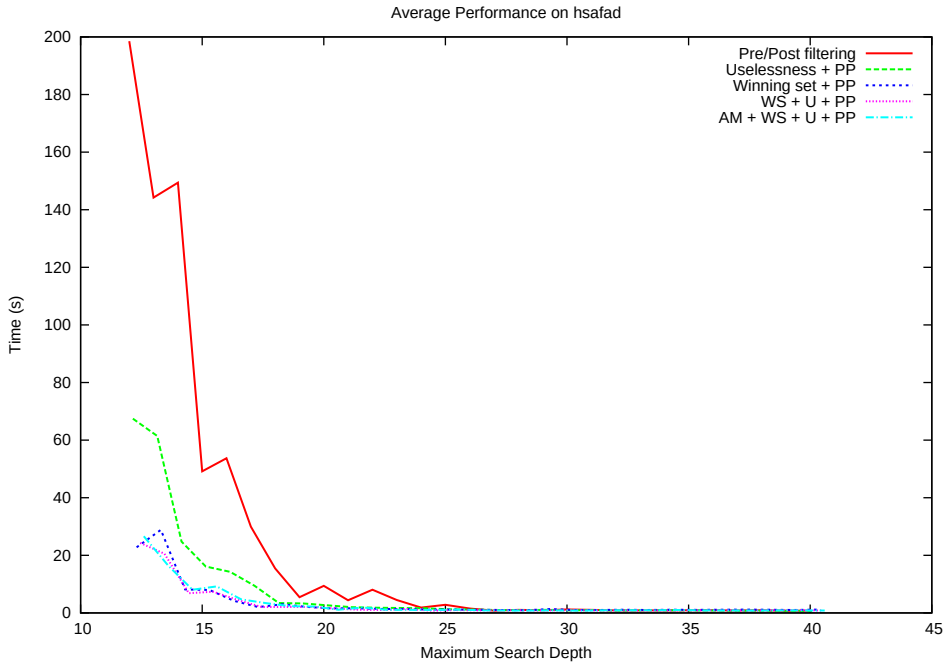


Figure 5: *Average performance on* HSAFAD *at various depths.*

4.2.1 Impact of optimizations

The impact of our optimizations varies significantly by benchmark, but shows clear, often dramatic improvement over our baseline: all of our benchmarks are analyzable within our time bounds at some depth, and in most cases, at least with multiple optimizations enabled, at all depths. The DPRYKH and RTZ CHAPTER 2 benchmarks pose the most significant challenges, mainly due to their large number of player commands (global in the case of DPRYKH, and local in the case of RTZ CHAPTER 2), and hence large branching factor. Even in these cases, however, optimization benefit is evident.

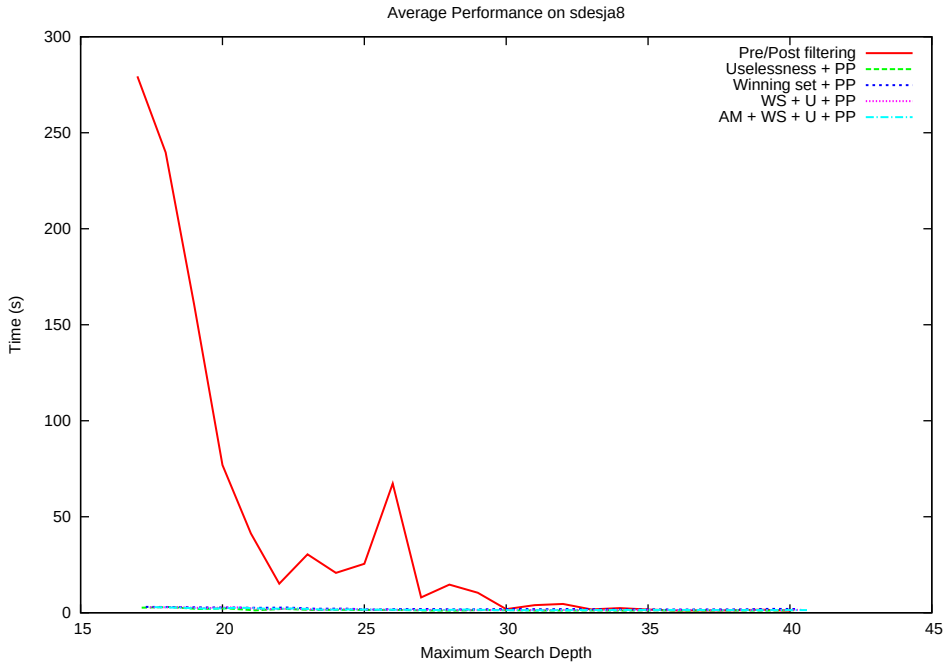


Figure 6: *Average performance on* SDESJA8 *at various depths.*

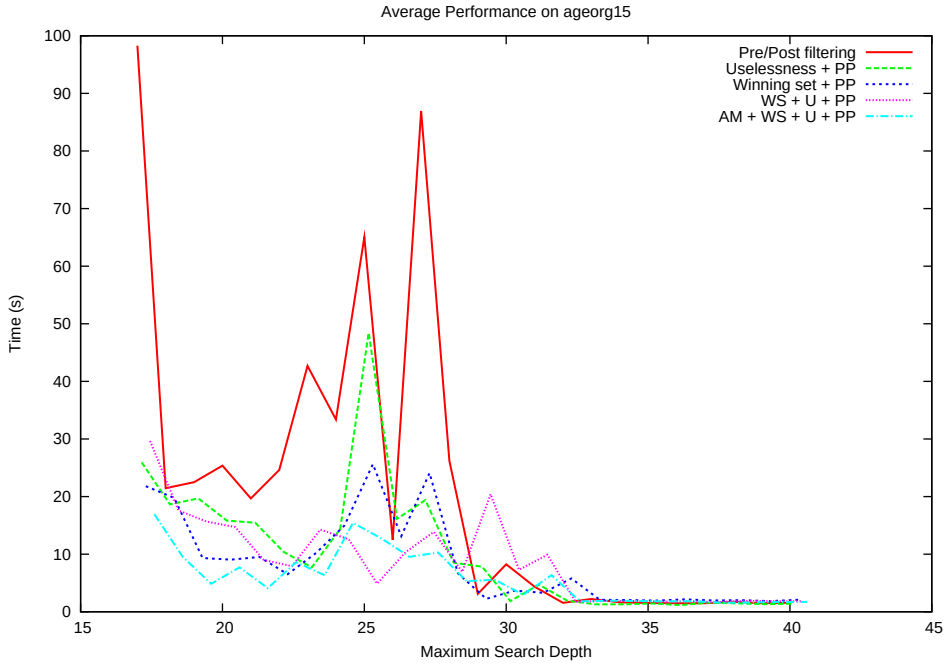


Figure 7: *Average performance on* AGEORG15 *at various depths.*

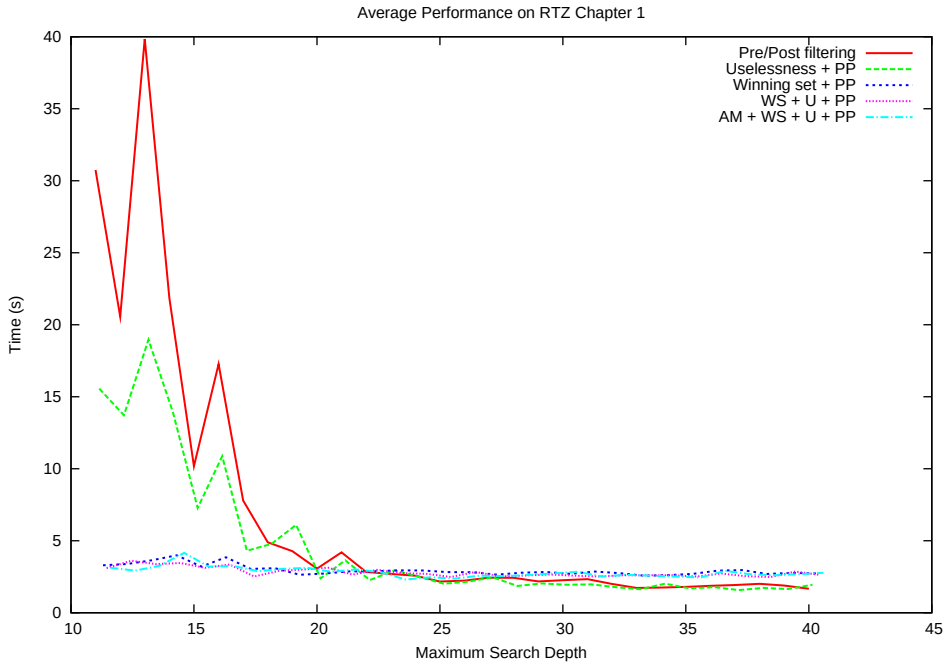


Figure 8: *Average performance on* RTZ CHAPTER 1 *at various depths.*

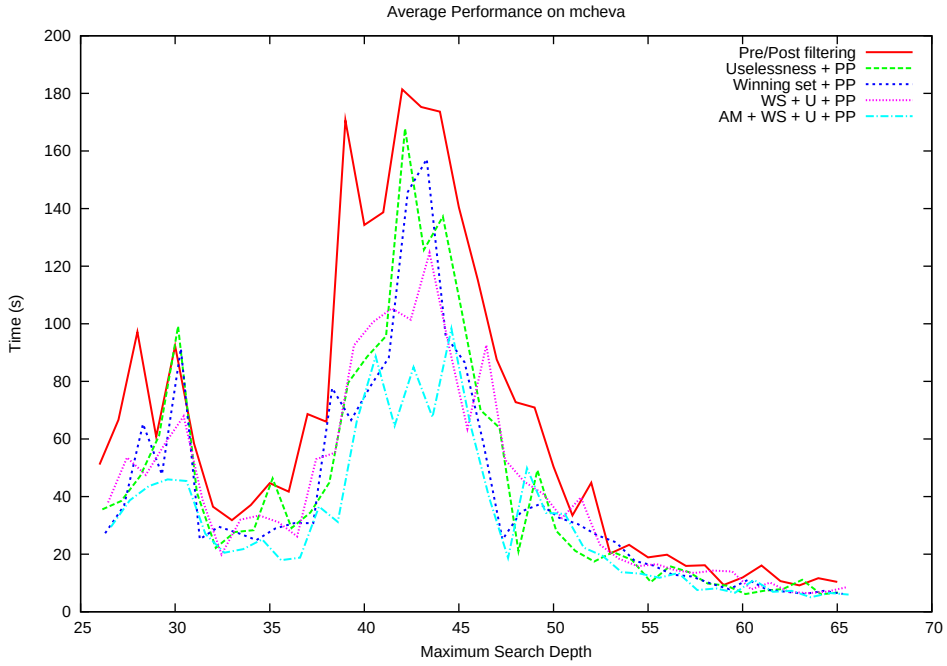


Figure 9: *Average performance on* MCHEVA *at various depths.*

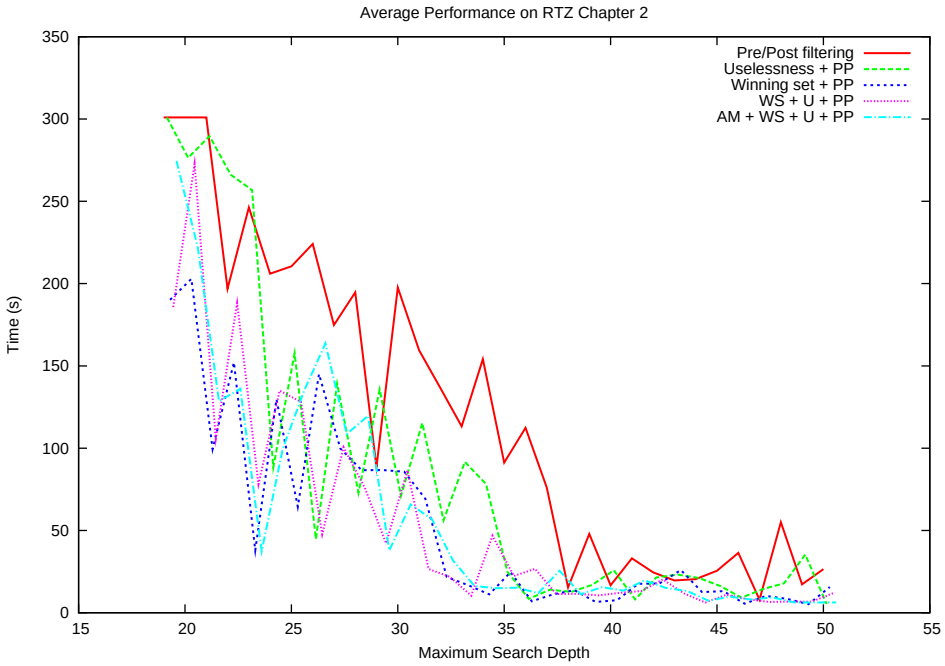


Figure 10: *Average performance on* RTZ CHAPTER 2 *at various depths.*

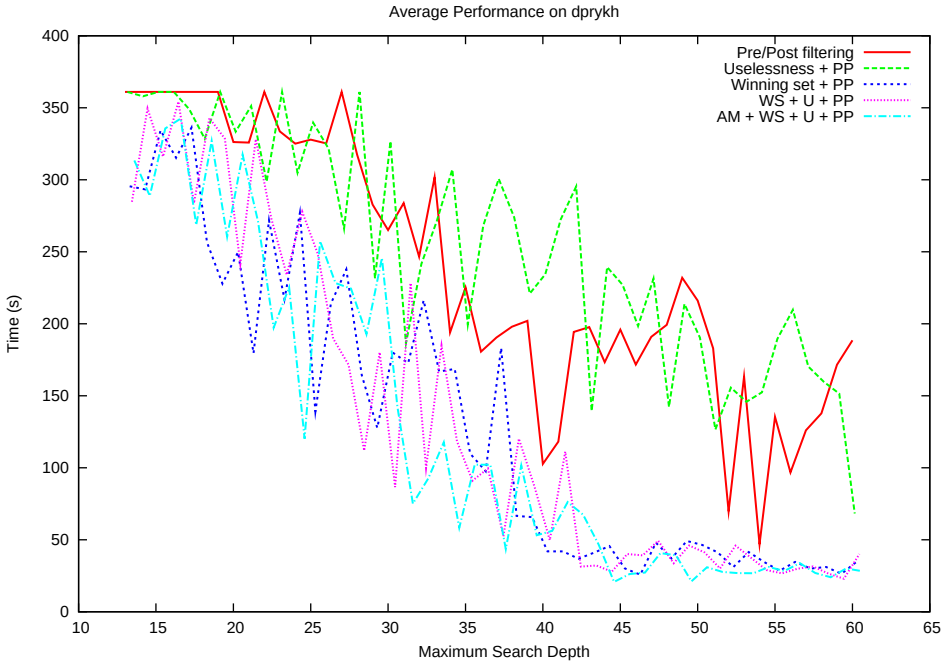


Figure 11: *Average performance on* DPRYKH *at various depths.*

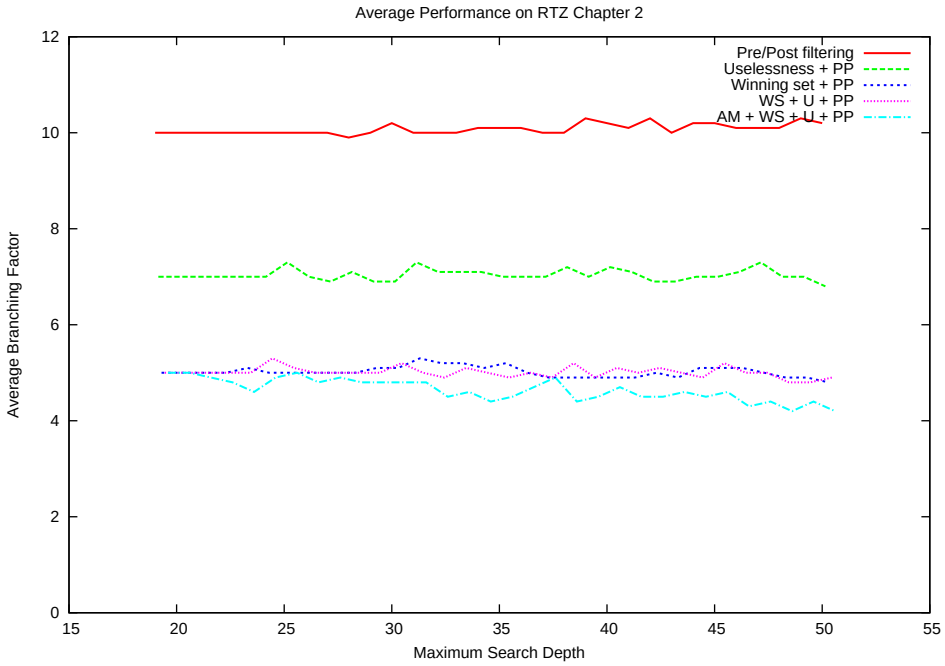


Figure 12: *Branching factor in* RTZ CHAPTER 2 *at various depths.*

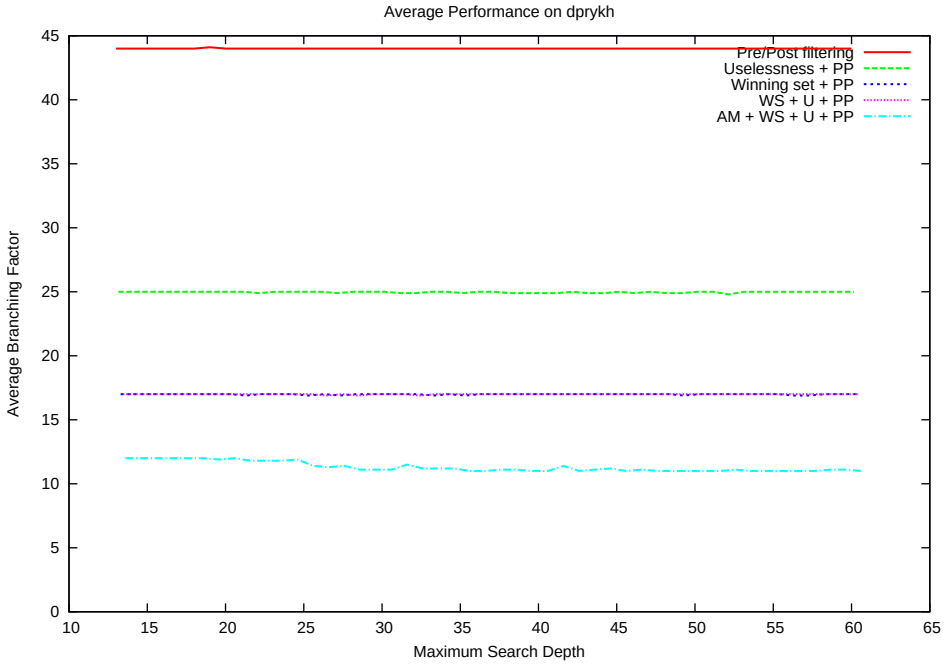


Figure 13: *Branching factor in* DPRYKH *at various depths.*

Most of the benefit in performance can be attributed to either the winning set or uselessness analysis, with the former usually showing greater improvement. As mentioned in Section 3.4, uselessness and winning set analyses overlap to some extent, and this can also be seen in our results—in most cases winning set and winning set + uselessness produce similar behaviour. This is reinforced by an examination of average branching factor; data for our two most branch-intensive benchmarks, RTZ Chapter 2 and dprykh are shown in Figures 12 and 13. Here we can see a quite stable impact from different optimizations, showing the winning set and uselessness analyses both help significantly, but do not tend to further improve behaviour when combined. From this we can also see that accurate match has a noticeable impact on branching factor. This is less obvious in our performance data, where variance and the fact that our other optimizations already improve the branching factor significantly reduce the apparent benefit.

4.2.2 Inherent winnability

A striking, and initially counter-intuitive feature of all our data sets is the way search times tend to decrease as maximum search depth is increased. Since the state space is exponential in the search depth, our expectation was that increasing depth would uniformly increase search time. Figures 5 through 11 show this is not the case, and we may even conclude that merely increasing search depth is a feasible search optimization strategy, if minimal solutions are not required.

This behaviour can be explained in relation to a few main factors. In a general sense finding a *minimum* length solution is a harder problem than just finding *any* solution. Minimum length solutions tend to be few, and the number of solution permutations grows very quickly if lattitude is given to contain extra sub-optimal actions or orderings. This may increase the relative solution density as depth grows, and patterns in solution densities can cause periodicity in the search performance as well, partially explaining the behaviour of mcheva (Figure 9) and ageorg15 (Figure 7). For smaller narratives, such as RTZ Chapter 1, our optimized search may also have reduced the state space to the point where it could be feasibly exhausted. Although this is unlikely in general, and certainly not true for larger games, we note that most IF games are both finite and designed to be eventually won. Narrative games are often time-intensive and have relatively low replayability—good game design avoids the need for the player to save and reload earlier states, and thus a player who plays the game without repetition and without losing should always be able to eventually win.

5 Related Work

Formal approaches to game narratives are driven by two major goals, analysis/verification, and interactive story generation. These share similar requirements, both needing to formally represent narratives, and both requiring some model of appropriate (or inappropriate) narrative properties.

Useful narrative properties center around identifying behaviours that players find pleasing, disruptive, or which may improve story generation. Adams, for instance, proposed basic guidelines for industry developers in a series of online articles [1], where among many other properties the problem of narrative game-play being allowed past the point of winnability was presented; Verbrugge formalized this notion as "pointlessness." [29]. Barros and Musse proposed a model designed to ensure pace/tension [6], and Nelson et al. defined properties related to the spatial locality of actions, as well as motivational or logical continuity of events [22]. Relevant concepts, such as measuring

minimum game solution length, are also found in multiplayer game analysis [10], and of course a variety of qualitative studies exist based on player studies [15]. Nelson and Mateas consider general state reachability issues as part of a high-level requirements analysis [21]. Our work here builds on previous experience in winnability analysis, applying a generic SAT-based solver to a Petri-net model of game narratives [23].

Narrative modeling and analysis is frequently approached from a logical perspective. Logical models can facilitate automated reasoning, allowing both story generation and analysis, and have been proposed in many formalisms. Kakas and Miller, for instance, define the $\mathcal{E}$ language for representing narratives, encoding knowledge of events and consequences in a system based on Event calculus [12]. An event-based logic is also used by *AIFT,* in a system that allows developers to write (non-trivial) IF games in a Prolog-like language [16], and Nelson and Mateas have proposed an event calculus model as an overall form of declarative game design [20]. Formal and logical contexts have been applied to an interesting variety of problems, including verifying the time-line of a non-linear story [9], ensuring logical correctness and continuity of events [14], and applying linear logic to analyze multiplayer game balance, competition, and scenario complexity [10]. In general narrative analysis is difficult, and Reiter argues that narrative complexity, including features such as recursion and non-determinism, makes showing narrative correctness as difficult as proving program correctness [25].

A number of authors have also investigated Petri-net models. Petri-nets are appealing for narrative representation due to their flexibility, naturally expressing non-determinism, repetition and even concurrency, as well as for their ability to easily model game resources. Verbrugge defined the *NFG* formalism, a structured, 1-safe Petri-net model of game narratives [29], and applied it modeling the inital scenes of a text-based adventure. Natkin and Vega define a set of basic Petri-net templates for representing logical connections between game events such as sequencing B before A, $B \Rightarrow \neg A$, *etc.,* and use it to represent portions of the game Myst [18]. A fairly complete, if less algorithmically derived Petri-net design is given for a simple exploration game by Araújo and Roque [2]. Colored Petri-net designs have also been described, for modeling a multiplayer commodity-trading game [24], and for controlling story development in a civic simulation [3]. In direct application, Petri-nets have also been used to model stories in the design/specification phase of the educational game "Europe 2045" [7]. Other, similar graph models are possible too; Szilas and Rety describe a custom graph model for representing stories, with the goal of describing minimal story structures that still maintain elements of drama and interactivity [27].

A formal approach to narrative has obvious applications to automatic story generation. Narratives have many repetitive and common elements, and several authors have described approaches based on assembling narratives from component structures [8]. Lang, for instance, proposed a declarative, rule-based system wherein axiomatic game events and non-terminals are mixed in grammar rules, and stories can be subsequently constructed through DFS search [13]. A central feature of story generation is the means by which important narrative properties such as temporal and logical continuity are ensured. Riedl and Young give an overview of interactive story generation approaches, arguing for mediation techniques where a centralized story manager analyzes and adapts narrative structure [26]. Barros and Musse analyze a number a number of planning algorithms/system, considering their appropriateness for AI-based, interactive storytelling [5]. Dória et al. propose using non-deterministic finite automata to represent behaviours, ensuring good behaviour through model checking [11], and Min et al. develop a hybrid story planning model, incorporating both dynamic, goal-based behaviour with more static story specifications based on anticipating and explicitly representing all narrative choices and paths [17].

In this work we *post-facto* analyze narratives manually developed as text-based, interactive fiction adventures. Although text-based IF games are an older genre, they have had as a whole a strong influence on more modern interactive fiction, adventure, and RPG game design. An active community continues to exist as well, and a number of IF authoring kits are available as products on their own [28, 30] or in terms of extensions built on older commercial offerings [19]. We have based our design on the PNFG language for interactive fiction [23]. PNFG has a very simple and well-defined syntax and semantics, simplifying analysis design. Future work for our approach includes extending it to current game genres and development environments.

6 Conclusions and Future Work

Narrative analysis has many potential applications. Here we have investigated and shown the practical feasibility of a basic reachability problem, but the techniques we use apply to a variety of interesting narrative verification questions: pre/post-condition analysis is relatively independent of goal, and winning set and uselessness are easily parameterized to focus on different goals. As well as solution length and complexity, properties more generically depending on game paths or state relations, such as "pointlessness" [29], or constraints on spatial locality [22] are search-related problems, and could be examined in our design. Further application is found in determining the path to more immediate game goals (such as how to open the next door), where search speed improvements are essential for developing online and interactive but generic and automated game hint systems.

We are currently working on extending our input model to accommodate other sources of game narratives, and in particular ones built into commercial RPG game engines. In these environments narrative description is not centralized, and interacts with general scripting environments as well as the graphical environment and user interface. This makes narrative extraction challenging, although our early experience is that it is also quite feasible.

References

[1] E. Adams. The designer's notebook: Bad game designer, no twinkie! parts I–VI. `http://www.gamasutra.com`, 1998–2005.

[2] M. Araújo and L. Roque. Modeling games with Petri nets. In A. Barry, K. Helen, and K. Tanya, editors, *Breaking New Ground: Innovation in Games, Play, Practice and Theory: Proceedings of the 2009 Digital Games Research Association Conference*, London, September 2009. Brunel University.

[3] D. Balas, C. Brom, A. Abonyi, and J. Gemrot. Hierarchical Petri nets for story plots featuring virtual humans. In *Proceedings fo the Foruth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 2–9, 2008.

[4] D. Barnett. Return to Zork. Activision Publishing, Inc., 1993.

[5] L. M. Barros and S. R. Musse. Planning algorithms for interactive storytelling. *Comput. Entertain.*, 5(1):4, 2007.

[6] L. M. Barros and S. R. Musse. Towards consistency in interactive storytelling: Tension arcs and dead-ends. *Comput. Entertain.*, 6(3):1–17, 2008.

[7] C. Brom, V. Sisler, and T. Holan. Story manager in 'Europe 2045' uses Petri nets. In *International Conference on Virtual Storytelling*, volume 4871 of *Lecture Notes in Computer Science*, pages 38–50. Springer, 2007.

[8] K. Brooks. Programming narrative. In *Proceedings of the 1997 IEEE Symposium on Visual Languages*, pages 380–386, 1997.

[9] J. Burg, A. Boyle, and S.-D. Lang. Using constraint logic programming to analyze the chronology in "A rose for Emily". *Computers and the Humanities*, 34(4):377–392, December 2000.

[10] F. Collé, R. Champagnat, and A. Prigent. Scenario analysis based on linear logic. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, page 1, New York, NY, USA, 2005. ACM.

[11] T. R. Dória, A. E. Ciarlini, and A. Andreatta. A nondeterministic model for controlling the dramatization of interactive stories. In *SRMC '08: Proceeding of the 2nd ACM international workshop on Story representation, mechanism and context*, pages 21–26, New York, NY, USA, 2008. ACM.

[12] A. C. Kakas and R. Miller. A simple declarative language for describing narratives with actions. *Journal of Logic Programming*, 31(1-3):157–200, 1997.

[13] R. R. Lang. A declarative model for simple narratives. In *Proceedings of the AAAI Fall Symposium on Narrative Intelligence*, pages 134–141. AAAI Press, 1999.

[14] C. A. Lindley and M. Eladhari. Causal normalisation: A methodology for coherent story logic design in computer role-playing games. In *CG'2002: International Conference on Computers and Games*, volume 2883 of *LNCS*, pages 292–307, July 2002.

[15] B. Mallon and B. Webb. Stand up and take your place: identifying narrative elements in narrative adventure and role-play games. *Computers in Entertainment (CIE)*, 3(1):6–26, 2005.

[16] D. Merritt. AIFT: Amzi! Interactive Fiction Toolkit. `http://www.ainewsletter.com/downloads/if_docs/`, July 2004.

[17] W.-H. Min, E.-S. Shim, Y.-J. Kim, and Y.-G. Cheong. Planning-integrated story graph for interactive narratives. In *SRMC '08: Proceeding of the 2nd ACM international workshop on Story representation, mechanism and context*, pages 27–32, New York, NY, USA, 2008. ACM.

[18] S. Natkin and L. Vega. A Petri net model for computer games analysis. *International Journal of Intelligent Games & Simulation*, 3(1):37–44, March 2004.

[19] G. Nelson, P. Seebach, R. Firth, A. Plotkin, and E. Short. Inform. `http://www.inform-fiction.org/`, 1993.

[20] M. J. Nelson and M. Mateas. Recombinable game mechanics for automated design support. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 84–89. The AAAI Press, 2008.

[21] M. J. Nelson and M. Mateas. A requirements analysis for videogame design support tools. In *FDG '09: Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 137–144, New York, NY, USA, 2009. ACM.

[22] M. J. Nelson, M. Mateas, D. L. Roberts, and C. L. I. Jr. Declarative optimization-based drama management in interactive fiction. *IEEE Computer Graphics and Applications*, 26(3):32–41, 2006.

[23] C. J. F. Pickett, C. Verbrugge, and F. Martineau. (P)NFG: A language and runtime system for structured computer narratives. In *Proceedings of the 1st Annual North American Game-On Conference (GameOn'NA 2005)*, pages 23–32. EUROSIS, Aug. 2005.

[24] M. K. Purvis. Narrative structures for multi-agent interaction. In *2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2004)*, pages 232–238, Beijing, China, Sept. 2004. IEEE Computer Society.

[25] R. Reiter. Narratives as programs. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *KR2000: Principles of Knowledge Representation and Reasoning*, pages 99–108, San Francisco, 2000. Morgan Kaufmann.

[26] M. O. Riedl and R. M. Young. From linear story generation to branching story graphs. *IEEE Comput. Graph. Appl.*, 26(3):23–31, 2006.

[27] N. Szilas and J.-H. Rety. Minimal structures for stories. In *SRMC '04: Proceedings of the 1st ACM workshop on Story representation, mechanism and context*, pages 25–32, New York, NY, USA, 2004. ACM.

[28] K. Tessman. *The Hugo Book—Hugo: An Interactive Fiction Design System*. The General Coffee Company Film Productions, Toronto, Canada, 1st edition, 2004. `http://www.generalcoffee.com`.

[29] C. Verbrugge. A structure for modern computer narratives. In *CG'2002: International Conference on Computers and Games*, volume 2883 of *LNCS*, pages 308–325, July 2002.

[30] C. Wild. *ADRIFT: Adventure Development & Runner—Interactive Fiction Toolkit, version 4.0 manual*, 2003. `http://www.adrift.org.uk`.