Design and Authorship for the Halberd Narrative Planning Engine

A Major Qualifying Project submitted to the faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the degree of
BACHELOR OF SCIENCE in
COMPUTER SCIENCE and
PROFESSIONAL WRITING and
INTERACTIVE MEDIA AND GAME DEVELOPMENT

Submitted by:
Richard Hayes
Myles Karam
Jonas McGowan-Martin
Brian Rubenstein
Gareth Solbeck

Advisors:
Professor Charles Rich (CS / IMGD)
Professor Mark Claypool (CS / IMGD)
Professor Ryan Madan (HUA)

Submitted:
April 27, 2017

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see http://www.wpi.edu/academics/ugradstudies/project-learning.html

Abstract

In interactive narratives, there is a trade-off between authorial control and player agency. To address this issue, our team developed Halberd, a game engine which uses planning techniques to produce branching narratives while keeping author workload manageable. Rather than writing story paths, the author writes a world, a set of characters, and an end goal, and the engine dynamically generates the story path based on player actions. Additionally, the engine handles procedural creation of scenes using individual art or text components. To evaluate the engine, we created and playtested a horror game, titled *The Worm of Everhill*. This report describes the design of the Halberd system, and discusses techniques of authorship for narrative planning engines.

Acknowledgments

We would like to thank the following people for their significant roles in our project:

- Cherish Springer, who created the art for *The Worm of Everhill*.
- Professor Ed Gutierrez, who advised Cherish Springer in creating the art.
- Professor Stephen G Ware, University of New Orleans, who wrote the Glaive Narrative Planner.

Table of Contents

List of Figures

1 Introduction

In interactive stories, there is a challenge called the "Narrative Paradox". The Narrative Paradox describes the challenge involved in reconciling the needs of the player, who is now a participant in the story, and the author, whose job is to provide a compelling story (Aylett, 2000). In games, the two extremes, full authorial control and full player control respectively, are linear games and sandbox games. While linear games guarantee the ability to portray an interesting narrative, they offer little in the way of player control. Sandbox games provide plenty of opportunity for the player to make choices, but are not guaranteed to provide a compelling story. In the middle, to reconcile the two extremes, are branching narratives. Branching narratives involve providing authored choices to the player, writing out every branch for each choice the player makes. The challenge with branching narratives, however, is the large amount of authorship necessary to make each branch compelling.

To attempt to reduce this increase in authorship load, programmers have used planning to try to procedurally generate narratives. Planning is a field of study within Computer Science that has been utilized by game developers for narrative design to attempt to procedurally generate narrative arcs, thus requiring that authors only write the world and goal (Young, Ware, Cassell, and Robertson, 2013). When applied to games, however, there is a problem with planning: it is a static process. This means that planning generates an entire plan, without accounting for player interaction. To address this problem, we created the Halberd Game Engine, which encapsulates a planning system called Glaive and provides methods for accepting player input and adjusting the generated plan in response to that input (Ware and Young, 2014). Furthermore, we developed a game, titled *The Worm of Everhill*, that utilizes the Halberd Engine. Halberd can be used to create both text-based and graphical games.

Overall, our experience with *The Worm of Everhill* and in creating the Halberd Game Engine gave us several insights with regards to two major areas: how to author interesting stories within a planning engine, and target areas for optimization within a planning system. Furthermore, we evaluated the success of *The Worm of Everhill* through playtesting to determine how interesting and enjoyable the game was to play.

Section two, the background section, provides an understanding of procedural content generation, writing techniques, and planning. This section also outlines some of the other approaches to using planning to attempt to create interactive stories. The third section discusses the design of the Halberd engine, breaking it down into three parts: the integration with Glaive, the text generation, and the graphics generation. Section four introduces *The Worm of Everhill*, and describes the process of making a game using Halberd. Section five outlines the findings of the project, including discoveries in designing stories for Halberd, optimizations made to the Glaive planning algorithm, and results from playtesting *The Worm of Everhill.* Finally, section six suggests some possibilities for future work in the field of interactive planning, both in Halberd and outside of the engine.

2 Background

Before discussing the technical details of the Halberd Engine, it is important to understand past work in areas relevant to our project. In this section, we describe how procedural content generation has been used in games. Next, the methods for writing compelling narratives in classical literature are compared and contrasted to how similar game narratives are approached by authors today. Lastly, we introduce the concept of planning in the context of artificial intelligence, and detail several of the challenges that must be overcome when applying this concept to games and narratives.

2.1 Procedural Content Generation

Procedural content generation (PCG) is "the algorithmic creation of content with limited or indirect user input" (Togelius, Shaker, and Nelson, 2016). Since the invention of this technology, PCG has been closely tied to the games industry. Its uses in games are twofold: PCG not only provides a way to add variation to a game between playthroughs, but also a way to add a seemingly infinite amount of content while keeping the storage requirements manageable. The works outlined in this section are early examples of PCG, and precursors for our work on the Halberd Engine.

It was the former property of PCG that led to its first implementation in a large-scale game in 1980 with the release of *Rogue*, a dungeon crawl game where the dungeon floors, enemies, and item drops are all randomly generated each time you play (Toy and Arnold, 2012). See Figure 2.1 for an example map. The developers were inspired by adventures from the Dungeons & Dragons role-playing game, and the very limited amount of single-player linear story computer games available at the time. Disappointed in the replayability of these adventure games, they created *Rogue*, which has since spawned an entire genre of games called "rogue-likes", which all share similar PCG techniques for procedurally generating the environment and level progression of the game each time you play.

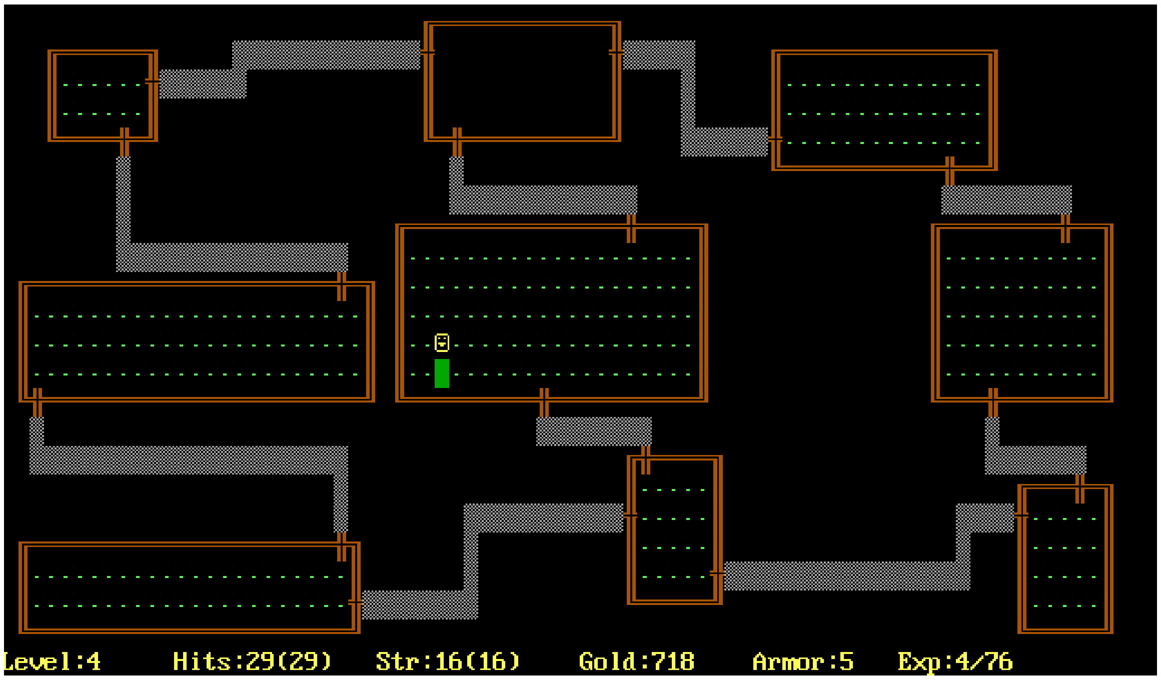


Figure 2.1: A screenshot of an algorithmically created map in *Rogue*

In 1984, another landmark game was released, titled *Elite*, that used PCG to keep storage requirements low (Bell, n.d.). Figure 2.2 shows a screenshot of this game. In *Elite*, players traveled around space to trade goods, make money, and upgrade their spaceships. However, the game's scope was limited by 8-bit computing. PCG was used in order to have enough planets for the game world to feel like it was on a galactic scale. The game used a seeded algorithm to generate eight galaxies with 256 planets each, allowing every planet to have unique properties. Since the game would only keep the nearby planets in memory and recompute others using the static algorithm when the player traveled, the game could be run within the memory constraints of 8-bit computers.

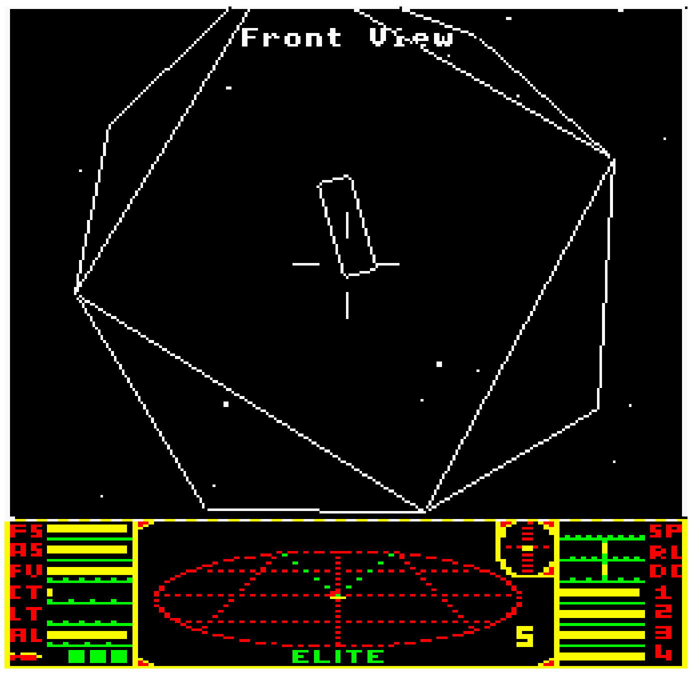


Figure 2.2: A screenshot of *Elite*, an early example of PCG

In both of these examples, PCG is used to generate the game environment. However, the possible uses of PCG are much more widespread. For example, procedurally generating text is particularly prominent in a genre of games called interactive fiction (IF). In an IF game, the player enters textual input and receives text as a response. This presents the challenge of converting the logical statements describing the game world into natural prose. Although many IF games, particularly early ones such as *Adventure* or *Zork*, are puzzle and mechanic focused, IF games in general have a more literary bent. As such, this makes them a strong example for procedural text generation (Montfort, 2005).

Façade, released in 2005, went beyond template-based procedural text generation to generate full sentences based on the context of the situation. This breakthrough work featured two non-player characters that the player could communicate with using typed sentences (Mateas and Stern, 2003). The player's actions influence what happens in the interactive story, and ultimately the outcome of the conversation between the player and NPC characters. *Façade* effectively procedurally generates the narrative of the game depending on player choices, and was one of the first pieces of interactive media to do this.

2.2 Writing

There are many different types of writing, and many different types of documents that can be authored. For the purpose of our project, we were mainly concerned with the field of creative writing and authoring practices used to write fiction. Examples of writing include books, journals, and other physical media. Physical literature dominated as the method for recording thoughts, stories, and history for a great many years, with the oldest known written story dating back to 2000 BC ("The Epic of Gilgamesh: The First Epic, from The First Civilization," n.d.). During this time, narrative and literary theories formed and have been extensively developed for studying and discussing the written story. Many of these theories are well accepted, but some of the concepts are reliant on the linear, unchanging nature of the medium for which they were first conceived. However, as modern technology has developed, new and more dynamic storytelling media – such as digital games – have begun being introduced. Since the goal of this project is to procedurally generate a narrative, the details of how to create a story by hand are of great interest to us. This section discusses some of the principles and techniques used for crafting stories in traditional literature, and how they translate into the world of game design.

2.2.1 Traditional Literature

While traditional linear literature differs in a lot of ways from the dynamic and interactive medium of digital games, there are techniques and guidelines for authoring a story that predate this medium and can be adopted into design to improve storytelling and the player's experience. While some traditional literature is purely about telling the reader the story of the characters in the authored world, many stories make an effort to cause the reader to feel close to, and empathize with, the main character. Some stories are even told in the first person, immersing the reader by making them a character in the world. The character may have a different name, different ideals, or different goals, but over time the reader suspends their reality to walk in the shoes of this character and feel like they are actually present in the story's events. Digital games frequently use the same techniques, placing the player in the role of the main character. Because of the interactive nature of digital games, immersion may be enhanced through the customization and personalization of a character, or by letting the player make choices for that character in-game. However, for all intents and purposes, the role of that character remains very similar to that of the main character in traditional literature.

As for the story itself, interactive or not, there are certain common elements that are said to be critical to the narrative's form. Janet Burroway, author of Writing Fiction: A Guide to Narrative Craft, claims that the three fundamental elements of fiction are conflict, crisis, and resolution. Conflict is paramount to fiction, Burroway says, because in fiction "only trouble is interesting." It is important to make the distinction that this is true of fiction and not of real life. A life where nothing ever goes wrong may be quite pleasant to live. However, it is unlikely to make a very captivating story. In contrast, even a minor conflict in a story has a way of drawing in the audience (Burroway, 1992)

Another important distinction from real life is that a conflict established in literature is guaranteed to come to an end. This is the reason that it is okay, and in many situations highly enjoyable, for a reader to read a violent, frightening or otherwise unpleasant event in a story: because that event is bound to come to a close. In literature, the events and conflicts are contained nicely in the words on the page. At the very least, a literary conflict will cease once the reader runs out of pages to read. That being said, a conflict ending due to a lack of pages is unlikely to be very satisfying. This is why crisis and resolution are important elements of a story. Following the establishment of a conflict, a reader expects that conflict to reach its climax and to ultimately be resolved. An important note is that the resolution to any conflict is not required to be a happy one (Burroway, 1992).

These defining features of a story – conflict, crisis, and resolution – are applicable to digital games as well as traditional literature. However, the implementation of these features may be more complicated depending on the type of game and how much narrative agency is offered to the player. Figure 2.3 below shows a common relative pacing for conflict, crisis, and resolution in traditional literature. In this narrative structure, the dramatic intensity of the narrative steadily increases over time, reaching a climax point near the end of the narrative. After the climax point, the intensity drops to provide a satisfying resolution. If the author of a novel chose to follow this pacing, it would mean they would simply need to successively write the events to the correct scale and in the proper order. The only obstacle would be their own writing skills and vision for the story. However, when interaction is introduced, the author must consider the fact that the player may not follow the exact pacing they intended and, if possible, may experience events in various orders. This and other complications have made storytelling through digital games into a challenge that game designers have been confronting for many years.

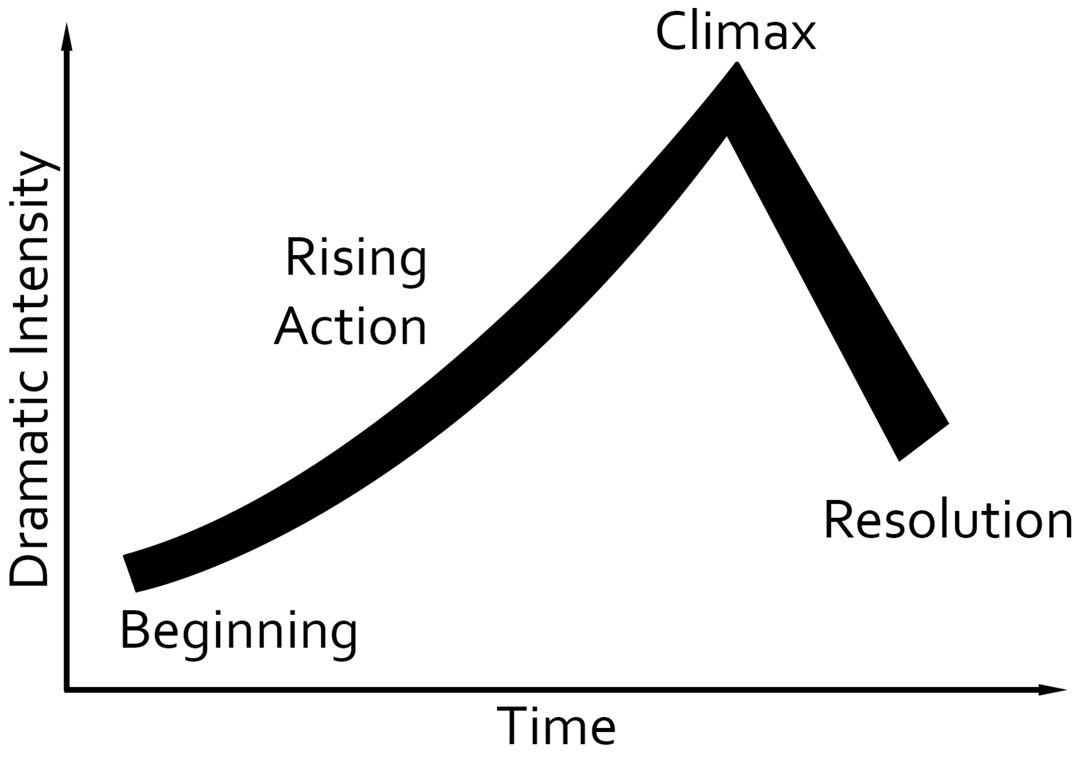


Figure 2.3: Common pacing for conflict, crisis, and resolution in traditional literature

2.2.2 Writing for Games

When writing narratives for games, authors must consider player agency in addition to plot structure. Aylett describes this as the "narrative paradox" of interactive narrative: reconciling the needs of a participating user with the need for overall narrative coherence. The author of a non-interactive story has complete control over the narrative, but this is not tenable in an interactive medium. A game completely controlled by the author limits the player to metaphorical page-turner. Instead, authors of interactive media must provide the player with agency, and allow them to participate in the narrative (Aylett, 2000).

Many games provide the player with minor agency, while defining the overall narrative as a single linear path. The player may choose how they overcome the next obstacle, but ultimately that obstacle must be overcome before the following one, and so on. The author writes a series of key events, connected by nonessential player-driven events, such as puzzles, combat, and the like. In such a narrative structure, an author can use define the exact progression of narrative events, in the same way as they would for a non-interactive narrative. This approach is pictured on the left of Figure 2.4.

On the other hand, games with branching narratives allow the players to make key decisions at various points throughout the story: to free the prisoner or leave them behind, to hunt down the werewolf or set a trap for it. Such games require that the author produce content for each branch of the narrative, but the author can still readily review all possible narrative paths. However, this approach does result in an increase in content needed with regard to player choices; the middle section of Figure 2.4 shows the increase in world states and transitions.

This increase in content has led to the development of various systems, such as simulations and narrative planners, to manage and produce content. However, such systems come with a reduction in the direct authorial control available. Rather than writing content directly, which produces a single experience or small set of experiences, the author tweaks the parameters of the system to adjust the range of possible experiences. For example, in a narrative simulation, an author might define one of the character's friends as having a low loyalty trait in lieu of directly writing a dramatic betrayal into the story. The rightmost section of Figure 2.4, illustrates this as a single world state and a set of transitions, which may be applied in any order to the initial world. The Halberd Game Engine described in this paper uses a narrative planner that dynamically creates a sequence of actions out of the ones provided, following this open ended story model to create a linear narrative.

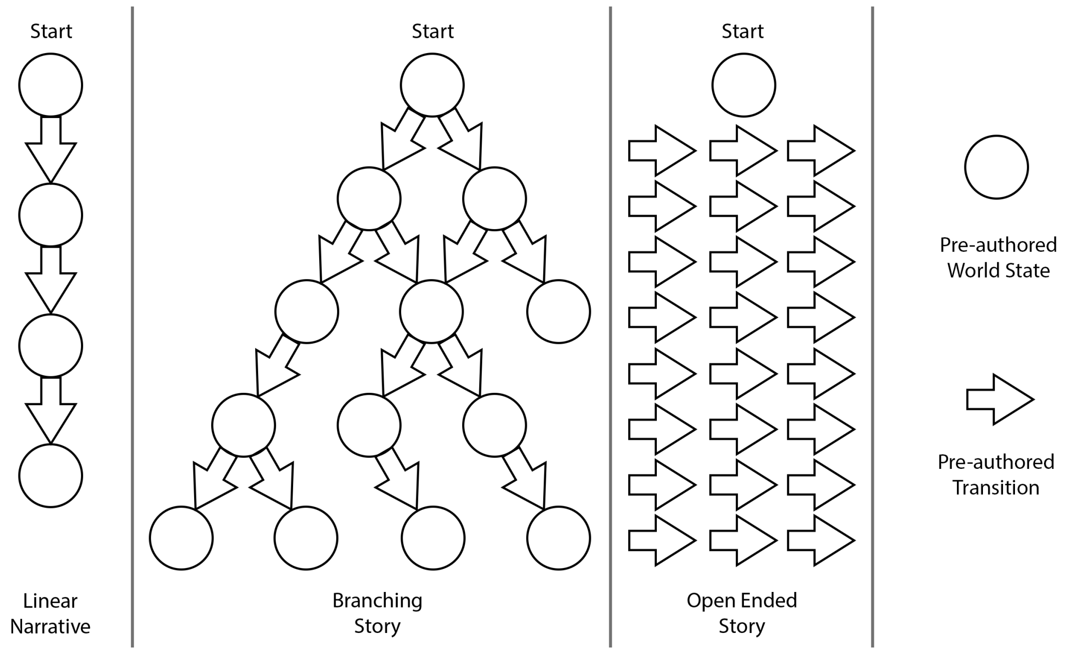


Figure 2.4: A comparison of authorship required for interactive story structures. Circles indicate the pre-authored world states and arrows indicate the pre-authored transitions between world states.

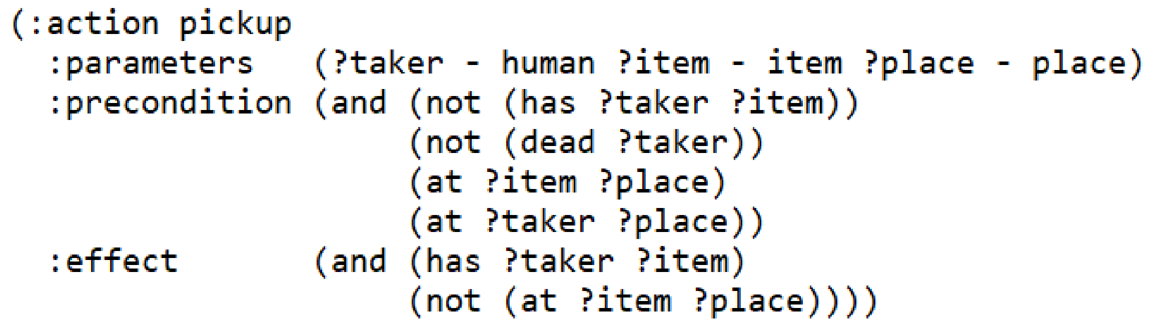
2.3 Planning

Planning is a traditional method of artificial intelligence (AI) which involves searching for a set of actions to transition from an initial state to a goal state. Over the last few decades, planning algorithms have been developed and refined in order to make them more feasible to be used in games. Among the first was the Stanford Research Institute Problem Solver (STRIPS) planner. After the development of the STRIPS planner, interest in planning algorithms has steadily grown, leading to the development of planning systems such as the Fast-Forward planning system, the Intent-Based Partial Order Causal Link planner, and the Glaive Narrative Planner (Hoffmann and Nebel, 2001) (Riedl and Young, 2010) (Ware and Young, 2014). The Halberd Game Engine uses the Glaive Narrative Planner as its core planning system, but research into other planners helped inform decisions about how to approach the development of the engine, and what work would be valuable to do within narrative generation moving forwards.

2.3.1 What is Planning?

Planning, also called automated planning, is a field of AI about determining a sequence of actions that transform a world state to satisfy some goal condition. At a basic level, finding this series of actions involves searching through possible sequences, one action at a time. One of the earlier planners, the STRIPS planner, used a formal language to describe planning problems in terms of first-order logic (Fikes and Nilsson, 1971). This language was later adapted into the Planning Domain Definition Language (PDDL), a semi-standardized language introduced to improve comparability of planning algorithms. PDDL is a list-like language, which describes world states as a set of predicates. Actions are described by planning operators, which specify first a set of parameters, and then the precondition and effect of the action in terms of those parameters. The precondition defines a logical statement in terms of predicates that must be true in order to apply the operator. The effect specifies the predicates which are added or removed when the operator is applied (McDermott et al., 1998).

In Figure 2.5 below, we show an example of both a PDDL operator and an instance of the operator. The operator, at the top, is what an author writes in PDDL syntax. Each operator has a set of typed parameters which it can be applied to. The `pickup` action applies to a human, an item, and a place. The preconditions restrict which combinations of parameters may be used: the human must not already have the item, the human must not be dead, and both the human and the item must be at the location. The truth of each of these conditions is indicated algorithmically by the presence or absence of the relevant predicate in the world state. The effects define the changes to the world state: once the `pickup` action has been used, the human now has the item, and the item is no longer at the place. Algorithmically, this is represented by the addition and removal of predicates in the world state (McDermott et al., 1998).



```
(:action pickup
  :parameters   (?taker - human ?item - item ?place - place)
  :precondition (and (not (has ?taker ?item))
                     (not (dead ?taker))
                     (at ?item ?place)
                     (at ?taker ?place))
  :effect       (and (has ?taker ?item)
                     (not (at ?item ?place))))
```

Figure 2.5: Example of PDDL action operator from *The Worm of Everhill*

PDDL also defines various extensions, such as including conditional clauses in the preconditions or effects of an action. These extensions improve the expressiveness of the language, at the cost of computational complexity (McDermott et al., 1998).

Until this point, we have described a plan as a sequence of actions. However, a subclass of planners called partial-order planners use a structure which describes the restrictions on relative ordering of actions rather than defining a linear total ordering of actions. Most typically, these are restrictions are represented using causal links, which connect an action's preconditions to the actions which establish each precondition. In the case of actions which depend on predicates in the initial state, the causal link is instead connected to a dummy action indicating this. This structure allows planners to require that some actions happen before others, without completely restraining the ordering (Young, 1999).

Many planners, particularly general planners, lack the processing efficiency to be used on large problems. Depending on the heuristic used in the search, the computational complexity can be such that planning for even short problems can take a prohibitively long time. Planners like the FF (Fast-Forward) Planning System have attempted to improve this search time. FF is a forward-searching state-space planner. Starting with the initial world state, it searches through possible successors (resultant states after having applied a single action) iteratively until reaching a state satisfying the goal. More specifically, FF uses a search heuristic called relaxed GRAPHPLAN, which estimates the distance of a state from the goal, in terms of the predicates present. FF also prunes the state space by identifying helpful actions and cutting branches where a subgoal is achieved too early. These methods result in a significant runtime improvement over other heuristic-search planners, and allow Halberd to re-plan without significant load times. (Hoffmann and Nebel, 2001).

2.3.2 Planning for Narratives

Many efforts have been made to model and generate narratives. The similarities between narrative structure and plan structure make planning systems a particularly appealing approach. In many models of narrative structure, the lowest level is called the fabula, and consists of the agents in a world, the actions carried out by these agents, and the relationships between these actions. Young points out the similarities between this and representational planning models, which also include agents, actions, and causal relationships. Narratives can be said to be plans, where the planning problem is constrained not only by logical and causal considerations, but also by narratological ones, such as conflict and character believability (Young, 1999).

Meehan's TALE-SPIN focuses on modeling the goals of characters. The goal of this system is to resolve the initial goals of the characters, given a set of intermediate subgoals and actions. As Meehan describes, a significant amount of effort is involved in codifying the knowledge of the world, both in the specification of possible actions and in the initial state of the world. This knowledge ranges from the physical constraints of the world to the emotional states and relationships of the characters. The consistency and believability of the narrative rely upon this knowledge (Meehan, 1977).

Later work by Dehn reflects upon TALE-SPIN, stating that its simulational approach is a less effective way of modelling a narrative. Her system, AUTHOR, focuses instead upon authorial intent, describing a story as the result of a series of author goals rather than character goals. As with the character goals in Meehan's system, these author goals are broken down into subgoals, which can then be resolved. Characters become tools for the author, brought into existence and justified when needed. Overall, AUTHOR models the authorial process of creating a narrative, but at the expense of character depth (Dehn, 1981).

Riedl & Young's planning algorithm IPOCL (Intentional Partial Order Causal Link) builds upon a more general class of planners called POCL planners. In general, these planners work by iteratively resolving flaws, such as unsatisfied preconditions, by adding actions until a complete plan is constructed. The IPOCL algorithm builds on this by annotating actions with the characters who must intend for that action to take place. The planner associates these actions to character goals, and requires that all actions in a plan have a goal association. This provides an overarching authorial goal for narrative guidance, and character goals to ensure that the actions taken are believable (Riedl and Young, 2010).

Later extension upon the IPOCL algorithm produced the Conflict Partial Order Causal Link algorithm. This algorithm uses threatened causal link flaws, which indicate where an action may prevent another's preconditions from being met, to model conflict. Actions may be included in the plan to justify intentions while ultimately being marked as non-executed, and thus not having any causal effect. This extension on IPOCL gives the planner more flexibility and allows it to model conflict, in which one or more characters are unable to complete their plan to reach some character goal

(Ware and Young, 2011).

Glaive, a planning algorithm by Ware, revisits the CPOCL algorithm's models of conflict and intentionality, but endeavors to find a balance between narrative modeling and algorithm runtime. Instead of a POCL planner, Glaive's core is a forward-searching state-space planner, which starts with a single node for the initial problem state and searches through a space constructed by applying actions to existing state nodes (Ware and Young, 2014). Glaive is the planning algorithm that powers Halberd due to its numerous runtime improvements.

2.3.3 Planning in Games

All of the algorithms discussed up to this point have been offline planners, which take a static problem definition and produce a valid plan. However, to utilize these tools in interactive narrative to address the narrative paradox, as described in Section 2.2.2, we need to consider the actions of the player. In many cases, player interactions may run counter to the intended narrative structure, and the system will need to resolve this in some way. In this section, we examine past uses of planning for narrative generation in games, and their approaches for incorporating player interaction.

LOGTELL is a tool for generating stories, which allows the user to work collaboratively with the system to author a narrative. Plots are generated iteratively, by alternately inferring new character goals and planning to satisfy those goals. After each planning phase, the user may either accept the given partial plot or ask the system to generate an alternative. Additionally, the user may give specific events or situations to be included in the narrative. These must be validated by the planner and may be rejected if no plan can be found to satisfy them. Because the user is always intervening indirectly, it is more permissible for the system to reject some user inputs; there is no immersion to be broken in this case (Karlsson, Ciarlini, Feijó, and Furtado, 2006).

Barber and Kudenko propose a system that limits the user's interactions to dilemmas. Each of these dilemmas gives the user a significant choice with two options, each of which is consistent with the narrative so far. For example, the Sacrifice dilemma gives the user the choice of whether or not to give something up in order to help a friend. Either of the choices will produce a coherent narrative, but the choices do clearly have a narrative impact. In this way, the user is still involved in the story, despite having only limited agency (Barber and Kudenko, 2007).

The narrative planning system Mimesis creates and maintains narratives as an AI controller in an existing virtual environment. First, it generates a story plan before the start of the game. Then, it directs NPC actions according to this plan, and mediates all user actions, checking whether they conflict with the constraints of the plan. If they do, the system may either intervene or accommodate the user. When intervening, the system may cause the action to fail, ideally in some world-consistent way. For example, the system may intervene when the player attempts to open a door by having the door be locked. Alternately, to accommodate the user, Mimesis must restructure the plan. This accommodation may be trivial (such as moving a future action to a new location) or much more substantive, and thus computationally expensive. Mimesis can estimate the computational cost of restructuring the system and determine which means of mediation it should use (Young et al., 2004).

3 The Halberd Engine

The goal for our project was to produce a game engine that uses an existing intentional planning algorithm at its core. We have designed our engine to work with Glaive in a way that allows player input. Since Glaive only runs one time and produces a plan that will bring the game from a defined start state to a defined end state, there is no method of interaction. To extend Glaive to include interaction, we implemented an engine named Halberd that integrates these individual static plans into a game loop. Halberd allows the player to interact with the planner, generating new plans that account for the player's choices.

Halberd consists of three components. The Game Manager is responsible for maintaining and updating the world state and narrative path. It interfaces with the Glaive narrative planner, which produces static partial-order plans as described previously in Section 2.3.1. The rendering system displays the world state and actions performed. Halberd provides two options for this rendering system: a text interface and a graphical interface.

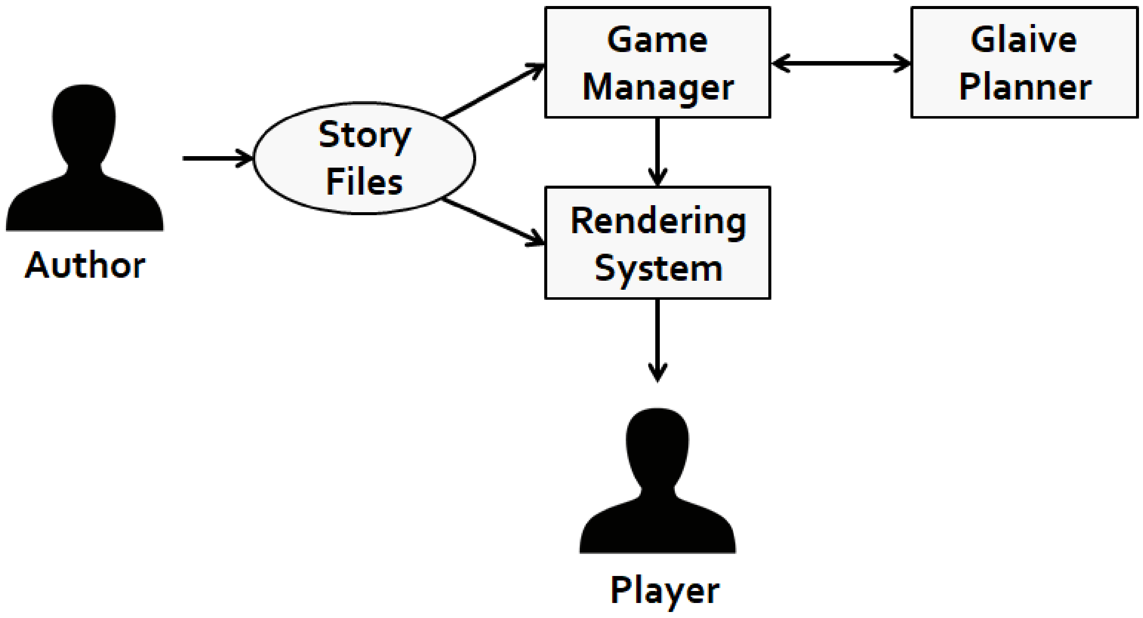


Figure 3.1: Architecture of the Halberd Game Engine

Halberd integrates the static planner into a game loop as follows. First, Halberd renders the starting world state. The player is then prompted to choose an action. Possible player actions are determined by comparing the preconditions defined in the planning problem to the current world state. After the player chooses an action, the planner generates a new plan, using the current world as an initial state. The game manager executes the first set of NPC actions from this plan. Finally, Halberd renders both the actions taken and the new world state, and then begin the loop again.

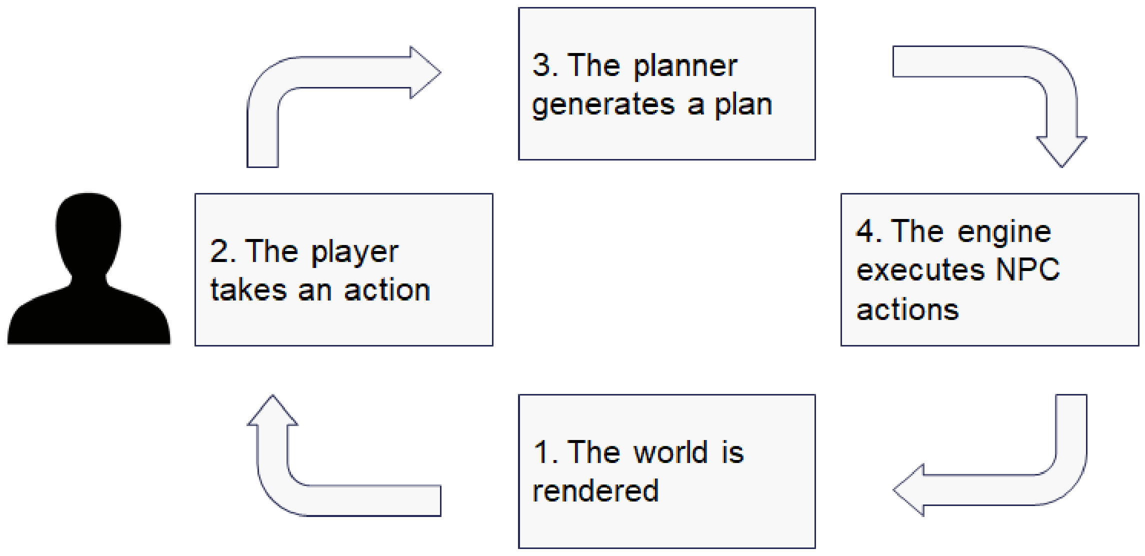

3. The planner generates a plan
2. The player takes an action
4. The engine executes NPC actions
1. The world is rendered

Figure 3.2: The Halberd game loop

3.1 Integration with the Glaive Narrative Planner

Integrating the existing Glaive algorithm, an algorithm made for single-time static use, into a game engine was one of the first challenges we faced. The integration required inputting new parameters into Glaive based on the results of past actions. The first step in converting Glaive into a dynamic system was to separate a subset of the plan generated, and execute those steps. Initially, Halberd made a single step per player action. We later implemented a system where each actor in the world could take one step per player action. However, this introduced some problems. In order for the defined plan to work, some characters needed to take specific actions at specific times. This meant that simply taking a single action for each actor was not a viable option. However, because Glaive is a partial-order planner, as described in Section 2.3.1, we were able to use the causal links that Glaive generates to determine which steps have had the necessary previous steps performed. After all these independent steps are executed, Halberd stores the new world state, and prompts the player for their next action based on the resulting world state. After the player's next action has been taken, Halberd replans from there.

Once the initial structure of integrating interaction into the planning algorithm was complete, we encountered a new complication within the system: state simplification. Glaive represents world states using a conjunction of logical predicates, as described previously in Section 2.3.1. When the state space is initially created, it provides every possible action and predicate, without accounting for what would actually makes sense for a character to do. Glaive provides a method of simplifying this state space to remove predicates and actions that are unintentional, and therefore would not be included in the story plan. This simplification decreases the planning time. However, due to the necessity of changes in character intention, we found that this could remove actions that, while not intentional at the start, would be necessary later. For example, in *The Worm of Everhill*, the talk-to-human and talk-to-possessed actions might be simplified away, but a player may later be interested in performing that action to gather information (See Appendices A and B for the complete domain and problem files for *The Worm of Everhill*). Furthermore, the player needs to be able to take actions that are not intentional with regards to the system's understanding of them. Therefore, the Halberd engine does not simplify the state space before running the game. See Section 5.2 for the runtime impacts of this change.

Once we had interaction working within Glaive, we created an API for Glaive, providing only specified functions to the higher level graphics and text rendering systems. The API hooks allow for creating a plan, pulling steps for specified characters out of the plan, making player and NPC steps, pulling information about the world, and getting the list of possible steps that can be made in the current world state. This encapsulation means that the text and graphical rendering within Halberd does not need to know about the planner.

3.2 Procedural Text Generation

At a basic level, the task of any game renderer is to present the relevant game state information to the player. Beyond that, the renderer is responsible for the way in which the player experiences the game. It is important that rendered content be clearly understandable and consistent, and provide an overall tone appropriate to the game.

As described in Section 3.1, Halberd tracks the world state using a set of logical predicates. While we could simply print these predicates out in standard PDDL format, this would not be easily readable, and would detract from the narrative. Instead, we chose to present the world using a series of descriptive sentences. In the text version, available actions were displayed as a numbered menu of imperative sentences, and executed actions were then displayed as descriptive sentences as well. In the graphical version, the actions available became clickable buttons, each of which contained the corresponding imperative sentence. To add more flavor to the text renderer, whenever the player moves to a new area, the renderer displays a paragraph description of the area.

To generate these sentences, Halberd uses a template-based approach. Actions and predicates are assigned phrase templates, with variables to be filled by objects. Objects are assigned phrases to be used in the larger templates. In this way, a large number of sentences can be produced from a relatively small number of phrases. This assembly is illustrated in Figure 3.3. Actions follow the same pattern, with two separate templates for the descriptive and imperative formats.

1. The predicate or action is extracted.
2. The template associated with the predicate or action is retrieved.
3. The required arguments used in the template are extracted.
4. The phrases associated with each argument are retrieved.
5. The phrases are assembled in the template.

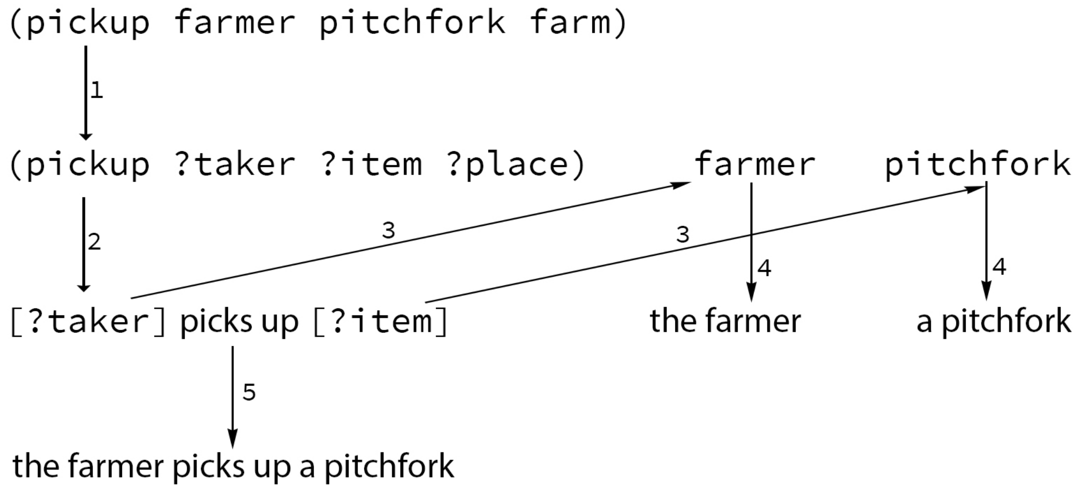

(pickup farmer pitchfork farm)
1
(pickup ?taker ?item ?place)
farmer
pitchfork
2
3
3
4
4
[?taker] picks up [?item]
the farmer
a pitchfork
5
the farmer picks up a pitchfork

Figure 3.3: The template assembly process, illustrated with the pickup action from *The Worm of Everhill*

This template-based approach alone is functional, but the output is often repetitive or stilted. To allow more variation, Halberd's text-rendering system supports adding multiple phrases for each object, action, or predicate. Whenever one of these needs to be rendered, the renderer chooses one of the associated phrases at random. In addition, authors may add specific phrases based on the arguments to the template. This is particularly useful for high-impact messages such as "`<character>` is dead" which would be more effective as phrases specifically tailored towards individual characters.

An example text rendering file can be found in Appendix C.

3.3 Procedural Graphics Generation

Similar to the way text is rendered in Halberd, the visual representations of the characters, places, and items in the world are all displayed based on the contents of the world state. The world state is first filtered for the predicates, characters, and items that are relevant to the player: specifically, those that should be viewable by the player in the game world. Then, through the decorations defined for the PDDL statements, images are drawn on the screen in the correct sizes and orientations. The design for the code and the art assets is covered in this section.

3.3.1 Design & Architecture

Unlike many traditional game engines, Halberd cannot use pre-authored environments for rendering, as scenes must be procedurally generated using components. We created a resource manager that loads these components and their positioning information upon launching the game. This decreases load times during a play session and simplifies the storage of these components. The resource manager also creates game objects for each of the primitives in the story. These objects are what the game logic uses to represent all of the nouns from the PDDL syntax and link them with the relevant graphics.

Once the resource manager has finished setup, play can begin. The game retrieves the player-filtered world state from the Glaive wrapper, and converts each of the PDDL predicates to an image, through string-matching the terms of the predicate to a game object. If the predicate has a game object that matches its terms, the game object is then added to the list of objects to be rendered. In the actual render function of the game screen, the objects are rendered according to their properties, which were the decorations loaded in through the resource manager. These properties depend on what type of game object is being rendered: location, character, or item.

Locations serve as the backdrop of a scene and provide guidelines on where to place other objects on top of them to make a well laid out image. They are decorated with slots to place characters and items on, oriented in spots on the location image where they would make sense. For example, if there is a table in the middle of a location image, the game object could be decorated with an object slot to have an item (if there are any items in the world state) rendered on the table. Character objects are decorated with the ability to hold items, defined by two pairs of points. Items are decorated with two corresponding points, which vary in placement from item to item. For a smaller item, the points are farther away from each other in order to scale it up to look natural when a character is holding it. The points may also be adjusted to change the angle at which a character holds an item. Defining these points in this manner allows the system to simply add an item to the character to hold. Orientation, mirroring, and scaling are all handled through the attachment points. See Figure 3.4 for a visual representation.

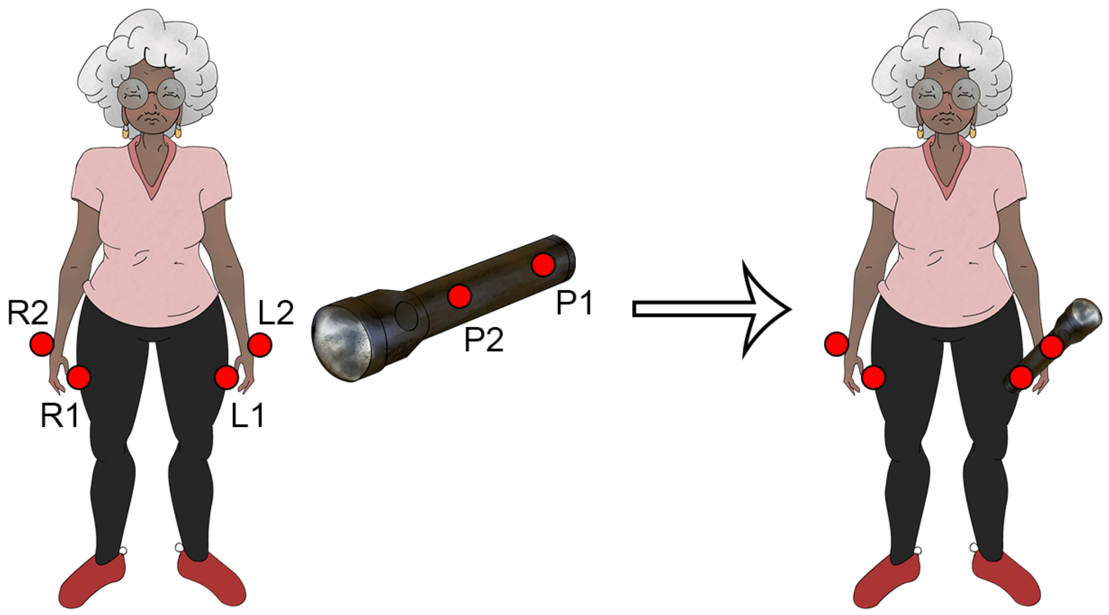


Figure 3.4: Image composition for characters holding items in Halberd, illustrated by art from *The Worm of Everhill*

3.3.2 Creating Art for Procedural Generation

As the code for the graphical system was developed, it became clear that art for a Halberd game had to be specifically designed for the system. Having dynamically changing world states means that the art needs to be dynamic as well. Locations must have open areas for characters and items to be laid out on, but must also not feel empty within those spaces because there are not always characters and items there. Characters must change visually depending on their state, defined by the world state. For example, in *The Worm of Everhill*, a different image is used for characters that are dead. Currently, Halberd supports PNG images. The resolution of the graphical panel is 800x1000 pixels. See Appendix E for examples of the art used in *The Worm of Everhill*.

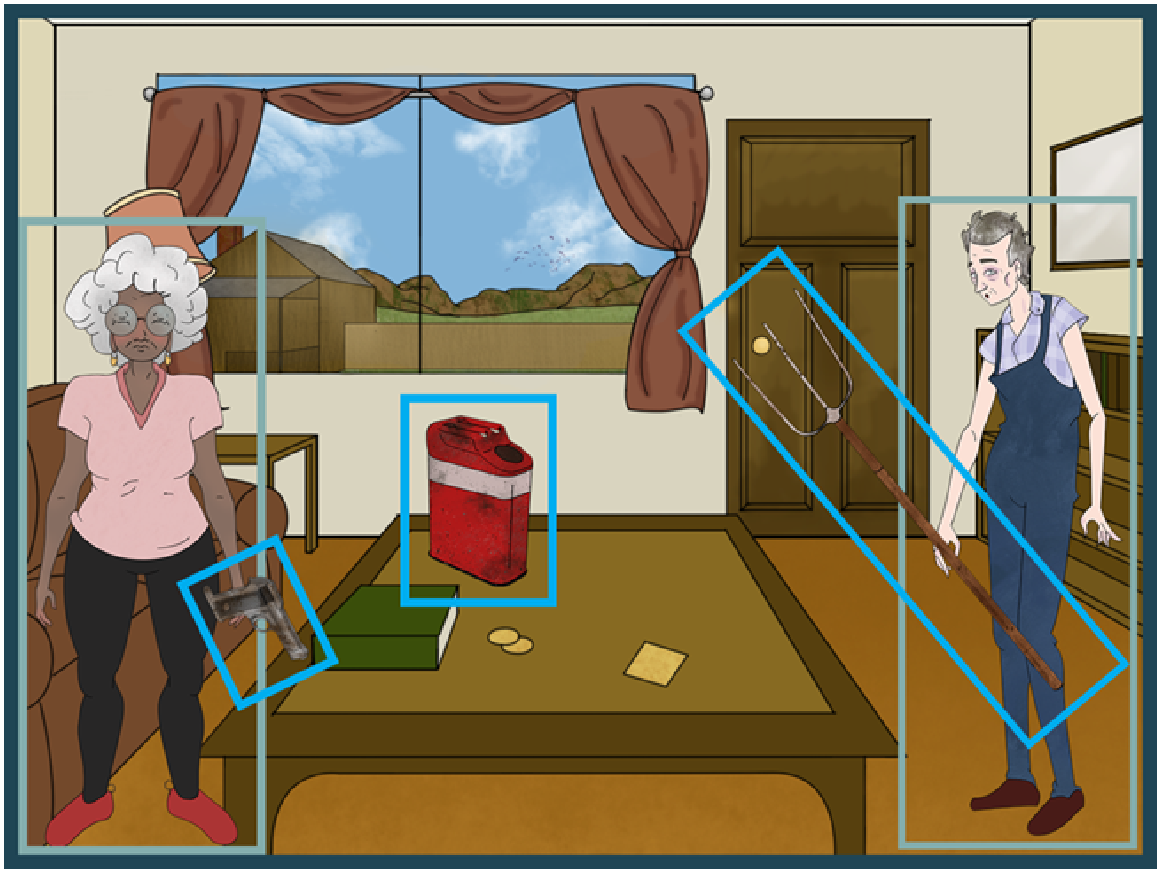


Figure 3.5: An example of component placement within the Halberd rendering system

4 Creating *The Worm of Everhill* with Halberd

To demonstrate the capabilities of Halberd, we authored a game titled *The Worm of Everhill*. In *The Worm of Everhill*, the sleepy, small town of Everhill is thrown into turmoil when a "mindworm" attacks. The mindworm is a small, slug-like creature that possesses a human, causing them to go on a murderous rampage. It is the player's job to find out who is possessed, and to stop the mindworm from killing everyone in the town, including the protagonist.

4.1 Genre Choice

This horror plot and setting was a good fit for how our planner reasons about conflict and intentions. In a typical horror story, there is an obvious conflict of interest between the forces of evil and the people just trying to stay alive or defeat the evil. The mindworm is the antagonist in *The Worm of Everhill*, and its intentions are to possess non-player characters (NPCs), and to make those NPCs kill the other characters. The conflict between these goals and the goals of all the other characters to stay alive demonstrates the conflict resolving capabilities of Halberd well.

The horror genre is also based on suspense, and a lot of *The Worm of Everhill's* story is generated behind the scenes from the player's point of view, building the suspense until the climax. Horror games often build atmosphere by showing the aftermath of certain "scary" or "bloody" events taking place, such as walking into a room with a dead body. In *The Worm of Everhill*, finding dead bodies is a large part of how suspense is built until the big reveal of who is currently possessed by the mindworm. The player not knowing a lot about the environment, including how the dead bodies got there, adds to this sense of suspense.

This hiding of information from the player to create tension was one of the main motivations for making a horror game in the Halberd Engine. Since player information is based on the world state at the current location, we made sure to include enough locations in order to have NPCs, especially the possessed one, take series of actions that are hidden from the player. This allows for body discovery, and walking in on incriminating situations for the mindworm-controlled NPC.

4.2 Actions

Even though horror games are based on suspense and the aftermath of certain events, there still needs to be some actions that the player can take that will allow for interesting gameplay. In *The Worm of Everhill*, we have implemented actions that are available to all humans in the game, including the player:

- Move from one location to another
- Pick up and put down objects

- Talk to human
- Douse human
- Ignite human
- Bind human
- Target human
- Kill human

These actions include the menial tasks of moving and managing the items a human is holding, and more dramatic events that lead to a human dying or being restrained. Some actions require the human doing them to have an item to do them with; for example, dousing requires a flammable liquid and igniting requires a lighter. It is important to note that even when the player manages to kill the human that is possessed by the mindworm, the game does not end, because the mindworm is not dead. The mindworm is still an active agent in the story, and according to its goals, it will slither off in search of a new host. The only way to defeat it for good is to light it on fire, which can be done whether it is in a human or not. These actions offer an interesting way to interact with the story environment.

4.3 Characters

Along with deciding actions that the player could perform, it was important to create a cast of interesting characters. In *The Worm of Everhill*, we evoked the sense of the horror genre through providing interesting characters and then having those characters turn against each other. Furthermore, we provided the player with a moral dilemma of what to do when one of the townspeople tried to kill them and evoked a feeling as though the player's life is on the line. The challenge was to create interesting characters without scripted narrative events to show character development.

To create these interesting characters, we scattered details about them through the graphics and text descriptions of the world. This provided a sense as though these characters were alive and active in this town. In *The Worm of Everhill*, there are eight different non-player characters, as follows:

- Matthias Cooper
- Jill Mills
- Jack Myers
- Joseph Mills
- Bruce Ableton

- Vivienne West
- Emilia Brooks
- William Archibald Barrington

We displayed characterizing details for each of these characters via their placement in the world and small graphical details. For example, Jack Myers was placed at the Worker's Respite, a bed & breakfast, in order to inform the player that they did not live in the town.

4.4 Items

For item design in *The Worm of Everhill*, we focused on making sure the items existed for the player to perform any actions that they were able to. Therefore, the game ended up having four different categories of items: weapons, combustibles, restraints, and igniters. Weapons allow characters to target and kill other characters. Similarly, combustibles allow dousing of other characters, restraints allow binding, and igniters allow ignition of doused characters. We chose how many of each of these types to place in the world based on the nature of the item type. For example, a restraint could only be used once, so we wanted to ensure that there were multiple restraints available. The final version of *The Worm of Everhill* includes the following items:

- Weapon: knife, gun, and pitchfork
- Combustible: gasoline and alcohol
- Restraint: handcuffs and duct tape
- Igniter: lighter

When choosing the items, we wanted to make sure that they made sense for the setting. Given that *The Worm of Everhill* takes place in a small town, we played into some of the small town stereotypes, such as including a pitchfork as one of the weapons. We were also careful to make sure that items were only placed at locations where it would make sense to find them.

4.5 Locations

Locations in *The Worm of Everhill* were designed to support the goal of creating interesting characters. For example, the Sheriff's Office location explicitly has a sign that reads "Mills for Mayor", to suggest to the player that there is an election in progress. Other locations, such as the Worker's Respite bed & breakfast, provide other details about the town. The locations were designed in a cohesive style, to create a consistent feeling of being in a small rural town. Overall, *The Worm of Everhill* had eight locations in total:

- the General Store
- the Town Hall
- the Player's Home
- the Sheriff's Office
- the Mills' Household
- the Worker's Respite
- the Hospital
- the Farm

These locations were chosen to provide all the basic locations that you may find in any given town. The general store provides a location which fulfills the need for commerce, and gives a sense of what the economy in this town might look like. The town hall and sheriff's office provide locations which represent government and authority. The player's home, the Mills' household, and the Worker's Respite provide locations to show what a house might look like in the town. The hospital and farm round out the locations, providing some other cliché small town locations to help make the world feel real.

4.6 Design Challenges

Something that we struggled with while designing the game world of Everhill was making the environment feel alive. The small town was supposed to be a very laid back place, but not devoid of any personality. The NPCs would stand in their starting locations and do nothing, and it was both unsettling and boring. Having characters do actions while idling is not something that the engine supports; actions must be explained by intentions. This meant that the challenge of NPCs standing and doing nothing could not be solved very easily with authoring, since the goal of the game and the intentions of the characters would need to be defined in such a way that the planner would want characters to act. Instead, to provide flavor to the world, the engine provides the ability to decorate the setting with text and graphics. In the text version, each location has a contextual paragraph that the player reads when they first arrive, which includes details about the setting and what the characters in it are doing. The graphical representations of characters and locations also try to inject as much life into the story as possible, with the artwork helping to provide life to the different characters. A subplot where the sheriff is running to replace the current mayor can be discovered by reading and paying attention to the campaign ads found in the location graphics.

Another issue that we ran into while writing *The Worm of Everhill* was that the player would unexpectedly die on the same step as moving to a new location. The reason for this was that the player would travel to a location where the mindworm-controlled NPC was. After the player's step to move, Halberd would have the possessed NPC take its step, which was to kill the player. To fix this, we added a targeting action; a human has to target another human before they can be killed. This allows for some counterplay when the player arrives in the same location as the mindworm-controlled human. If the mindworm-controlled human targets the player, the player can quickly escape and try again.

5 Results

Our goal for this project was to explore the use of a planning algorithm as the core of a game engine. Doing so would allow us to use planning to reduce author workload when creating branching storyline games. To begin our project, we studied the Glaive planner, and then designed the Halberd Engine that uses Glaive as its main algorithm. During the creation of Halberd, we created *The Worm of Everhill*, a game world that is specifically designed to use a planner in order to run. We have learned a lot over the course of our work with Halberd and with *The Worm of Everhill* with respect to story design and game design using a planner. This section discusses how story design using a planner differs from conventional story design as well as the feasibility of using a planner in a game setting. Finally, we playtested our game to get some feedback on both how well our engine works and how well our story was designed.

5.1 Story Design

While developing and working with the Halberd Engine, one of the goals we set for ourselves was to develop strong practices for constructing interactive stories within Halberd. Not only did these stories need to work within the technical boundaries of the software itself, but they had to be interesting and compelling for the player. The procedural nature of the story generation meant that an author would not be able to pre-write every detail in the story, or the order in which the scenes would happen. Player agency meant that the story would most likely deviate from the planner's initial plan. However, it was still important to have strategies that would give the game developer or story author control over the feeling of their story, and to address experience goals, story length, game endings, and story scope. Additionally, we felt it was important to write meaningful and believable characters in order to get the player truly invested in a story. This section describes some of the challenges encountered while designing plan-based stories, and the ways that we adapted to these challenges.

5.1.1 Meaningful Characters

One issue we noticed in early development of *The Worm of Everhill* is that there is a tendency to create one-dimensional characters when designing in a planning engine. In the story's logic files, the author of the story is only given one word to describe a character, which will serve as a label for them. However, knowing more than a simple label for these characters can often help make the world feel more real, even if very little of the written background will actually make it into the game. Furthermore, this character information can help with smaller, seemingly unrelated design decisions.

Originally, in The Worm of Everhill, each character was simply defined by a profession, or a first name. This meant that playing through the game felt very dry, and it was unlikely that the player would feel any attachment to the characters. In an attempt to address this, we wrote out names, descriptions, and small pieces of information, such as hobbies. We used this to both inform our placement of items, and provide information to our artist to use as a reference. For example, when the character Jill became Jill Mills, the young woman in her early twenties who aspires to become a mechanical engineer, we realized that a logical place for the gasoline item we already had in the game was at her house. In addition, this allowed our artist to use this information to help with drawing, not only Jill, but also the Mills' Household.

With regards to text rendering, we also used written character information to work small details into the descriptions for text-variants and locations to make the world feel more real. One example of this was the relationship between Mayor Ableton and Sheriff Mills. We determined early in our world-building that there was a mayoral election coming up, and that these two characters were competing for the position of Mayor of Everhill. Therefore, in the text description for the Sheriff's Office, we show the player that the front desk has several pins that have "Mills for Mayor" printed on them. Similarly, when we provide the death description for Mayor Ableton, we describe an "Ableton for Mayor" pin laying at his side, stained with blood. While we do not explicitly state that there is a mayoral election in progress, these little details can give the player the impression that they are interacting with a realistic world, even if they do not get to see it all. This helps offset the lack of scripted events and prose which are used to add characterization in more directly authored game narratives.

5.1.2 Designing Goals

In a traditional PDDL architecture, the goal that the planner is planning for is defined in the problem file. This is acceptable for a non-interactive plan and simple interactive stories that move toward a single end-state, but we encountered some problems with using this structure for our design. Having a single goal that the planner was working to meet was technically acceptable, but the player was capable of performing actions that prevented the planner from finding a valid path to that goal. In such a case, the planner could no longer take any valid steps and the game would end. This result would likely be both jarring and unsatisfying to the typical player. While we could include multiple acceptable goals, this caused the planner to take much longer to find a solution because it was searching for a valid path to each goal. This, again, would likely be acceptable for non-interactive plan generation as the planner would only need to run once and a player would not be waiting to take an action, but was far from ideal for our purposes.

Our solution was to abstract the goal out of the problem and to create logic for multiple goals in the domain. We did this by creating a single predicate, for example "goal-achieved," and having the goal defined in the problem simply be that the predicate is active in the world. Then, in the problem, we created logic in multiple axioms (one for each possible goal we wanted in the story) that would activate the goal predicate under certain conditions. This allowed us to create numerous options for acceptable end-states, and it also helped us avoid the story awkwardly ending because the planner could not find a plan.

Another function of the Halberd Engine is to provide a concept of location that can be checked before certain details are revealed to the player. This function introduces the concept of having "on-stage" and "off-stage" actions. If a tree falls down in the woods and the player's character is off in the city somewhere playing an arcade game, it does not necessarily make sense for the player to know it happened. This is an optional feature, but we made use of it in The Worm of Everhill. Instead of informing the player who the mindworm possesses and when it happens, and when and where each person in the game is killed, these actions can happen off-stage and will only be told to the player if the player's character is in the same location at the time the actions take place. This adds a sense of mystery to the game, allowing the player to determine for themself that a possessed character is behaving strangely, or to find the dead body of a character who was earlier killed off-stage. Unfortunately, this also created a problem: if the goal of the story was reached by actions that took place off-stage, the game would end and the player would not know why.

This problem highlighted an important aspect of the goal-authoring process for these plan-based stories: if authors want to avoid the story concluding without their knowledge, they need to include the player character as a necessary part of the goal. For The Worm of Everhill, this meant that the goal for the mindworm should not just be to travel around and kill all of the non-player characters in the town, but to also hunt the player's character. Another acceptable goal was for the player to kill the mindworm, saving the town (or at least whatever was left of it). This particular goal would not be something that the planner would plan towards, as the planner could not take actions for the player, but it was important to recognize this goal if it occurred so that the game could end appropriately.

5.1.3 Scope of Story

The scope of the stories we could create with our engine was an authorial concern constrained by the limitations of the technology we were using. In terms of the technical limitations, we were working within the time and memory constraints of the computer we were playing the game on. Authorially, there was no limit to the number of characters, predicates, items, locations, actions, axioms and goals that we could include in the PDDLs. In theory, we could have crafted an epic tale with thousands of characters that took place in many locations all around our fictional game world. In practice, every addition to the PDDLs created extra logical relations, actions and possible paths that the planner had to consider, and the higher the number of additions, the slower the planner was to find a plan. As we needed to replan after each action the player took, this was a constraint we had to take very seriously. Another significant constraint was the computer's memory. Some additions or logic in the PDDL caused the planner to use more memory space and sometimes even throw an error because it ran out of memory. Even if the player was willing to wait for minutes at a time between each action, they would not be able to continue playing the game if the system ran out of memory.

In a scenario where an author is working with an ideal computer that can always find a plan instantly and has infinite memory space, there are still authorial considerations to be made about scope. If an author of one of these plan-based stories decides to use the on-stage, off-stage features that were discussed in the previous section, then a very large world with many characters means that there is a very high probability that the player will not be where the most exciting actions are taking place. In a story where the author has chosen not to filter information by location, a world with many characters and many locations would mean that an equally large amount of information about the world-state would need to be provided to the player either through art or text. In general, both technically and authorially a better choice is to keep the number of characters, locations, and items in a plan-based story to a low number, if possible. For example, *The Worm of Everhill* includes only ten characters, eight items, and eight locations.

5.2 Runtime & Optimization

As our story grew in scope, the runtime of the Glaive algorithm grew. This forced us to be thoughtful about the content included in our stories, and to examine possible optimizations of the Glaive algorithm in the context of the Halberd Engine. We found that there were two places in particular which were important to examine: the startup phase of our story, and the time between actions, or search phase. The setup phase is run once, at the start of the game, whereas the search stage is run each time the player takes an action. In the startup phase, the majority of the runtime problem came from creating the search space, that is, the set of all possible actions, predicates, and characters. Once the game had already started, the wait-time between different actions was a result of the search stage.

We decided to approach the search phase first, in order to reduce the wait time between actions. On brief examination, we determined that approximately nine-tenths of the runtime in this stage was being used to construct individual actors' goal trees. Therefore, we explored the idea of storing these goal trees between runs. However, with the state changed, the goal trees that the Glaive Planning Algorithm created were no longer valid. This meant that the stored trees ended up not being useful. This led us to our second approach: multithreading. When creating goal trees for the individual agents, we noticed that each goal tree was being created independently of the other goal trees. Therefore, we explored spinning off a thread to generate the goal tree for each of the agents. If there are no actions which have more than one agent, the threads are independent, making this approach a viable option.

In order to examine the runtime of the Halberd engine before and after our algorithm modifications, we ran the Glaive algorithm on the initial planning problem for *The Worm of Everhill*, and measured the lengths of the setup phase and search phase. We used four different versions of the algorithm: with and without our threading modifications, and with and without simplifying the state space before searching. These benchmarks were run on a Windows 7 computer, with Java version 1.8.0, and a 4-core processor. Figure 5.1 shows the average results from 100 runs of these tests. In the unsimplified planning algorithm, the threading modifications provided a 66.4% runtime decrease within the search phase for *The Worm of Everhill*, while leaving the setup phase relatively unaffected.

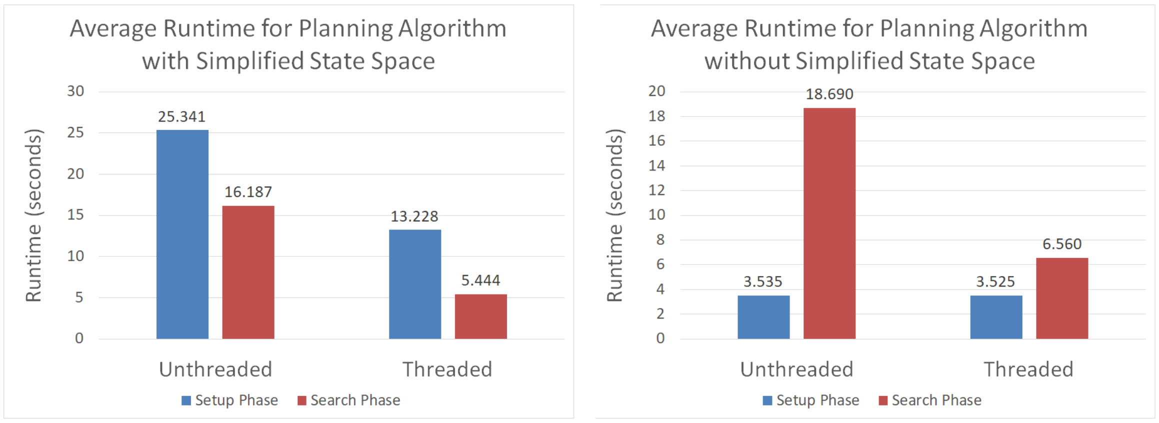


Figure 5.1: Average runtime for two phases of the Glaive planning algorithm, with and without threading in the goal tree construction process

Regarding the setup phase, we discovered an issue with delegation in narrative plans. Delegation is a method of describing what happens when one actor tells another actor to accomplish a goal for them. In Glaive, delegation expands the state space significantly, increasing memory usage to the point that it was not feasible for use in our game engine. Glaive uses a system of initially creating a fully detailed state space that extends the domain file by creating all possible actions and predicates from the set of objects in the problem. When delegation is included, this can create an exponential explosion of the state space, as Glaive's parsing algorithm does not differentiate which objects can delegate, and which objects cannot. Furthermore, Glaive does not differentiate which objects can be delegated to. This means that, until it simplifies the world state, any object can delegate any predicate to any other object. This gives it a $O(n^2m)$ space complexity, where n represents all objects, and m represents all predicates. Because the predicates are also based on the number of objects, this sort of brute-force delegation approach is not feasible for large-scale game development within our engine.

5.3 Playtesting

In order to to evaluate the Halberd Engine and our game, *The Worm of Everhill*, we carried out formal playtests with WPI students. Our playtest methods, followed by results, are explained in this chapter. Overall, each student was allotted a 30-minute time period to play our game, while two members of our group observed and took notes. After the player completed the game, they were surveyed to see how much they enjoyed the game, and if they would play it again.

5.3.1 Methods for Playtesting

To begin, we emailed all Computer Science and Interactive Media and Game Development students at WPI to inform them of our playtesting. The email included a short description of our game and a schedule in which they could each sign up for a 30-minute play session. We then followed the same steps, listed below, for each participant:

1. The participant was given a consent form (See Appendix F), informing them of why they were being playtested, and the risks and benefits of participating in our playtest.
2. The participant was given a short introduction to our game (See Appendix G), read aloud by one of the facilitators.
3. The participant was then allowed to play the game. During this playthrough, the facilitators recorded the reactions of the participant while playing.
4. At the end of the playtest, the participant was given a survey with a series of questions, which included how much they liked the game, what its strengths and weaknesses were, and whether they would play it again. Our complete list of survey questions can be found in Appendix H.

5.3.2 Results of Playtesting

Towards the end of our project, we carried out formative playtesting. The formative playtesting was during one of our beta versions of our game, and we had a total of 10 playtesters. During this playtesting, we were able to discover any issues with either our game or engine, and made fixes immediately. This section will discuss the feedback received from the playtests.

To begin, our game was rated with a difficulty of intermediate to hard. One participant mentioned "that you can die very easily" which is what we expected, but participants took this as having to play more than once to keep learning about the game. When asked whether the game consistently held their interest, the response of our playtesters were generally positive. Figure 5.2 summarizes these responses.

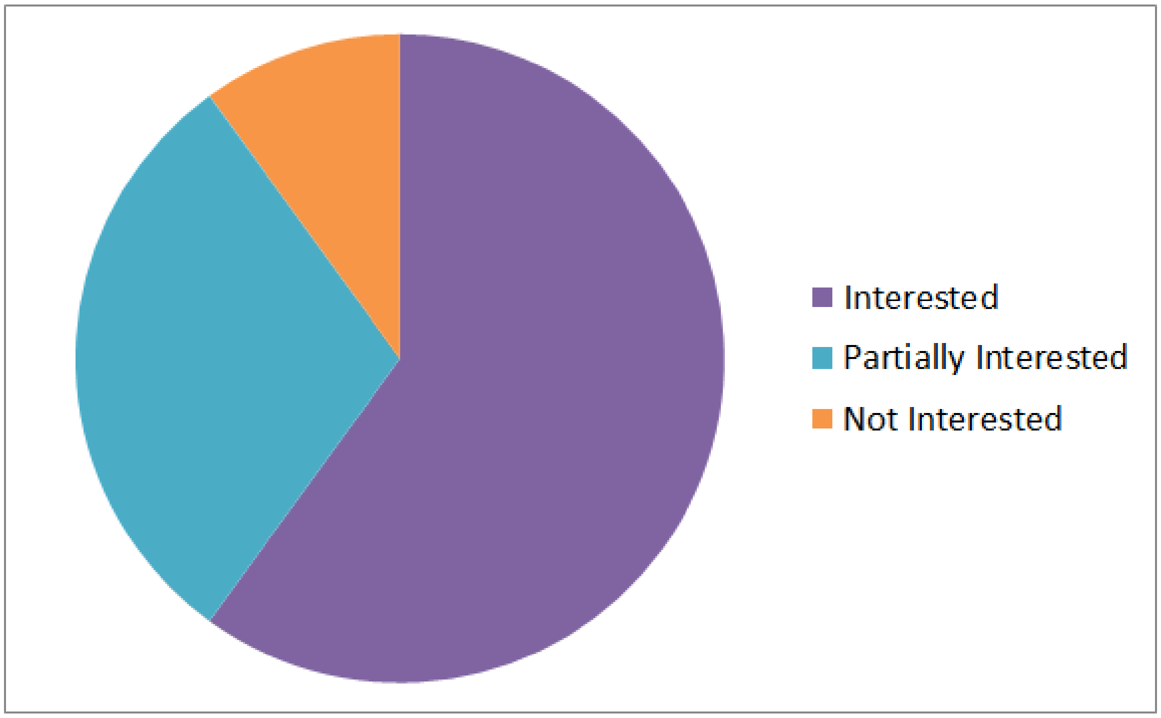


Figure 5.2: A graph showing participants' self-reported interest during playtesting

Based on these responses, it is safe to say that many of the participants enjoyed playing *The Worm of Everhill*, especially since many of them did actually play more than once.

Besides difficulty, we also questioned the participants on how they would describe our game. We were happy to see many participants described it as a horror-esque text adventure game with graphics, because this was our original intention. One participant even said "horror text adventure. It reminded me of X-Files".

Finally, we asked how well the participants understood the user interface (UI) of the game. Overall, our UI was between easy and intermediate to understand, as shown by the results in Figure 5.3.

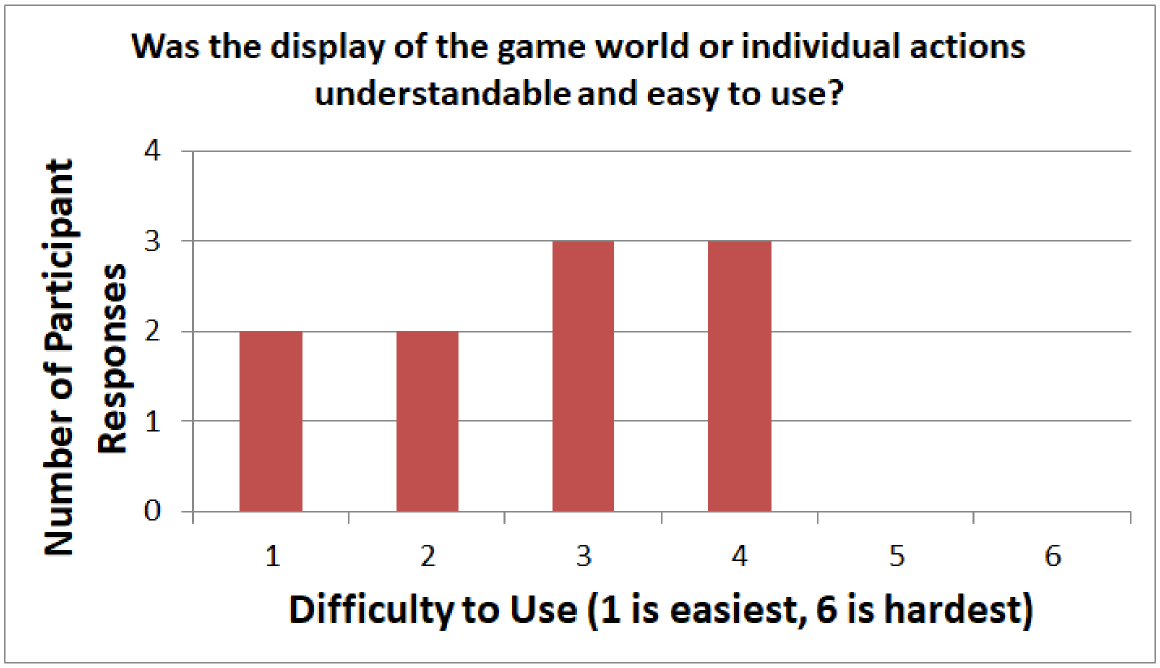


Figure 5.3: Survey results about UI understandability

The biggest issues with the UI that were pointed out to us were how we displayed possible player actions, and the amount of text displayed. Many individuals said it was too cluttered, so we organized the buttons into a scrollable section. We also removed a portion the text displayed, giving the UI a more organized and cleaner look.

6 Future Work

There is considerable effort to be done in the field of planning before it reaches the level of commercially authored branching storyline games. Further algorithmic improvements to reduce runtime and space usage within planning systems will provide the ability to create not only broader, more expansive worlds, but also longer stories. With current hardware capabilities, we recommend further exploration into more intentional agents within fewer locations. One of the weak points of *The Worm of Everhill* was that there was a lack of intentional agents, the mindworm and its possessed human being the only non-player characters acting within the game. Games that take place within fewer locations, with lots of characters taking steps, lend themselves to the strengths of planning more than a setting with few intentional characters such as *The Worm of Everhill*.

There are also several algorithmic improvements that should be investigated in regards to interactive planning specifically. One of the challenges with utilizing the capabilities of the Glaive Narrative Planner in an interactive context was making use of the simplified state space. A simplification algorithm developed specifically with an interactive narrative system in mind could help reduce the loading time between steps. Characters that need to act outside the bounds of their intentions could be flagged, such that actions or predicates involving them would not be simplified away. Furthermore, similar flags could help reduce the space complexity of delegation, noting which characters can delegate, and which can be delegated to. This could provide increased functionality to authors, giving them better tools to write complex worlds for interactive planning.

7 Conclusion

Frequently, game designers will approach narrative in a similar way to linear stories. However, because of the interactive nature of games, this approach is not very effective. A tension called the “narrative paradox” exists between the author’s direct control of a narrative and the player’s ability to influence that narrative. Various systems and approaches have been developed in the past to generate narratives in order to find a compromise between the two.

To explore planning as an approach to narrative generation, we developed the Halberd Narrative Planning Engine. Halberd uses the Glaive narrative planner to direct game narrative. Supported by other procedural content generation techniques, Halberd provides a platform for procedurally generated, turn-based narrative games. Our game, *The Worm of Everhill*, showcases the techniques used in this engine.

When authoring for a planning game engine, the authorship focus shifts from defining narrative paths to defining more about the world and characters. By focusing on creating a detailed world, authors can achieve the narrative development that more linear engines might afford through scripted events. Furthermore, the definition of end states allows the author to provide satisfying endings for the player of the game.

We have determined that planning currently provides the ability to create dynamic short stories, with high replay value. Algorithmic improvements and greater investigation into authoring approaches within a planning system will greatly improve the quality of stories that can be generated. Within a planning engine, authors create worlds rather than stories, and watch as the narrative is generated, coming to life before their players’ eyes.

Supplement: Authoring Games for the Halberd Narrative Planning Engine

Richard Hayes

Abstract

This goal of this project was to create a game engine that facilitated the design and authorship of interactive branching narratives in games, addressing the problem of the heavy workload associated with these narratives. Additionally, this study looks into the best practices for authoring a story using Halberd, highlighting the benefits and constraints of the engine, and discussing the project team's experience when writing *The Worm of Everhill*: a planning-based horror game created with Halberd.

1 Introduction

There exists a desire among storytellers to welcome participation into a narrative: to allow the audience to interact with a story and even change its outcome. We see this from authors who write Choose Your Own Adventure books, from actors and comedians who improvise on audience suggestions, and from game developers who work to blend the act of play and storytelling into rule-based environments. However, there seems to be difficulties that are intrinsic to this task, particularly when improvisation is not an option.

One such difficulty is this: the more agency that an audience has in a story, the more difficult it becomes to maintain a coherent narrative. This is known as the 'narrative paradox', and refers to the balance between the need of the author to provide a satisfying story structure, and the need of the audience to interactively affect the story and their experience. This is often seen as a scale (Aylett, 2000). On one end of the scale, too much audience agency results in a narrative becoming more of a conversation, a toy, or a simulation, depending on the medium. At the other end, we have traditional literature, movies, plays, and other stories in which the text that makes up the narrative remains the same each time they are told. It is possible to balance this scale, but is the middle-ground between these two extremes a desirable form of storytelling?

A second difficulty with incorporating audience agency into a narrative is the branching problem. This problem arises when improvisation is not an option, such as in written works and digital games. In these situations, the written text or code is unable to simply make up a response to the audience the same way that an improvising actor could. Every time that audience agency is introduced in these mediums, a suitable response must be authored for every change the audience can make. If the audience can choose to make one change out of two options, a story-path must be authored to account for either choice. If the audience is again offered a similar choice after the first one, an author must now account for four story-paths. The two story-paths split into four after the second choice, and a third would split those four into eight. This exponential increase in an author's workload can quickly become unmanageable.

These difficulties are what our team wanted to address with this project. As game developers, we could not rely on human improvisation to address changes made by the audience on these types of stories. However, we were unsatisfied by the storytelling practices we saw in many games that simply acted as ways to avoid or work around this issue, such as short story-paths that quickly ended a story, or decisions that would only temporarily change the course of a narrative before returning to a single path. We wanted our audience, the player, to truly be a part of the story, to make decisions that had meaningful impacts on the narrative and not simply be tricked by the illusion of choice. However, we also wanted the narrative goals of our stories and the dramatic or emotional experiences that we design to remain intact, and we couldn't simply put in the time to manually author an exponentially increasing number of story-paths.

To address these needs, we developed the Halberd Narrative Planning Engine: a game engine that uses planning software to facilitate the design and authorship of interactive, branching narratives. Halberd was made to shift an author's focus away from painstakingly authoring an extra story-path for every choice that the player makes, and towards story goals, character development and world building. Halberd also blends player actions and story progression, dynamically re-planning a story's events to accommodate the player's choices. In this paper, I discuss how the engine accomplishes these tasks. I will also suggest the best practices for authoring a story in this engine, while referring to some of the lessons my group learned when writing our own game: *The Worm of Everhill*.

2 The Halberd Narrative Planning Engine

In this section, I will discuss some of the technical aspects of the Halberd Narrative Planning Engine that provide context both for the project and for the authoring techniques that will be discussed later in the chapter: Authoring Plan-Based Stories for Halberd.

Halberd uses a technique called planning to generate story-paths on the fly, as the player of a game makes choices that affect the game world. In order to best utilize the engine and to make informed decisions when authoring a story for Halberd, it is important to understand some of the technical context for this project, including the benefits and constraints of the software.

2.1 Glaive and PDDL

As a starting point for the project, our team obtained permission to alter a pre-existing planning software called Glaive, that was developed by Stephen Ware for his dissertation work on automated narrative planning at the University of New Orleans. What differentiates Glaive from a classical planner is its ability to construct a plan based on intentionality. This means that a character can be written into an authored world with some sort of intention or character goal, and Glaive will incorporate that goal into its generated plan. This made the planner an attractive foundation for a project focused on procedurally generating interactive narratives (Ware and Young, 2014).

Choosing Glaive as a starting point meant that authoring would require the use of the Planning Domain Definition Language, or PDDL, which is a semi-standardized planning language used in Artificial Intelligence. For discussion purposes, there are several concepts related to PDDL that will be important to authoring stories. Firstly, two PDDL files are used to generate a narrative: a domain file and a problem file. In a domain file, the author specifies the types of things that will be in their world, predicates that apply to those things, actions that can be taken, and axioms that affect the world. By "types of things", I am referring to categories that the elements of the world fall under. For example, many stories include people, places and objects. In a domain file, an author doesn't get specific about who the people will be; they just say that "people" will exist in this world. Instead of "people", a fantasy story might want to say that "humans", "elves", and "goblins" exist in the world. Predicates essentially refer to adjectives that may apply to the things in the world: a place may be dark, a person may be armed or hungry, or an object may be cold. Actions are what the planner uses to generate plans. Examples of actions could include a character moving from one place to another, two characters having a conversation, a character picking up an item, or a character getting their hair cut. Finally, axioms are logical rules for the authored world, usually to apply a predicate under some circumstance; if a character is holding a weapon then they should be considered armed, or if a character is in the same area as a monster then that character should be frightened.

The second PDDL file, the problem file, is a lot simpler. The problem is where the author specifies the specific instances of the things that they defined in the domain. For example, if the author stated in the domain that people exist in the world, the problem would be where the author would say who the specific people are: “Mary Jane”, “John Smith”, etc. Secondly, the author uses the problem file to define the beginning state of the world. For example, John is at the coffee shop, Mary is at the bus stop, John is hungry, etc. The problem is also where the author can specify individual character goals, such as saying that John intends to get engaged to Mary, and also a final goal for the story, such as “Mary and John are engaged and Mary got her dream job.” The planner will take the initial world state from the problem file and, using the actions and axioms from the domain file, generate a story-path from that initial state to a state in which the goals are satisfied.

2.2 How Glaive Makes Choices

In order to generate story-paths from an initial state defined in the problem file and actions and rules defined in the domain file, Glaive considers every possible story-path that both succeeds at getting to the goal defined in the problem, and “makes sense” from a planning standpoint. What I mean by “makes sense” here, is that a story-path will not be considered if it includes a character acting against their own intentional goals. Additionally, Glaive will not allow for circular logic. Technically, a story-path would still get to the goal if a character picked up and put down a book many times in a row before starting the sequence of actions that lead to the goal, but as the number of times this happens could be expanded infinitely and does not facilitate the advancement of the story, paths like this are not considered.

Not only will Glaive avoid paths where a character acts against their own intentions, but it will attempt to satisfy character goals defined by these intentions while generating a path to the story goal. When defining an action that characters can take in the domain file, an author can specify one or more intentional agents for that action, essentially stating that the characters involved in that action must consent to the action in order for it to occur. Glaive uses these definitions to decide what actions are possible for each character based on their intentions.

Finally, after examining all of the potential paths to take a story from the initial state to the goal state, Glaive will always choose the shortest story-path available. This is typically ideal for a planner to do, because a shorter path to a solution for a technical problem is more efficient. However, this is potentially a weakness of using a planner for narrative generation. The shortest possible story is not necessarily the most satisfying. It was out of the scope for our project to change this aspect of Glaive, but its presence meant that there were extra considerations to be made when authoring a story for the Halberd Engine. These considerations are discussed in the chapter: Authoring Plan-Based Stories for Halberd, under the section titled

"Conceptual Challenge: Rigidity."

2.3 Displaying Text to the Player

PDDL uses a syntax that is made to be easily parsed by planning software. While this syntax is human-readable, particularly to one who is comfortable with the language, it is decidedly different from how we typically structure our sentences when speaking or writing. The story-path output by Glaive is produced in a similar syntax. This is ideal for the code in the software, but it presented a problem when it came to authoring stories with Halberd: we wanted to present the text for the story in the format that readers would expect to see.

The translation of the narrative text from PDDL representation to English sentences was not too difficult in concept. If Halberd created a state where a character named "Joe" was holding a cup of coffee, the PDDL representation would look something like: `(holding joe coffee)`. Essentially what we wanted to do was to have an English sentence prepared to use in place of that representation any time it was encountered in a story-path. What we wanted to avoid, however, was recreating the branching narrative problem where we would be manually writing a replacement sentence for every possible combination of elements that the planner might give us in PDDL form.

To address this issue, our team introduced a third file for the planner to use alongside the domain and problems files: a text file written in the JavaScript Object Notation (JSON). This file was where we would include manually written replacement text as described above, but with an advantage that JSON provided to us: the use of Java-style formatted strings. For our purposes, these formatted strings gave us the ability to write fill-in-the-blank style sentences and fill in those blanks programmatically. To go back to our example of `(holding joe coffee)`, we could manually author a single sentence for the predicate "holding" using format specifiers that act as placeholders for the elements related to that predicate: "`%1$s` is holding `%2$s`". Halberd could then fill in those blanks to create the sentence "`joe` is holding `coffee`." Furthermore, the JSON file allowed us to provide aliases for elements in the PDDL. While the PDDL uses the representation `joe` in its logic, we could use the JSON to automatically replace `joe` with "Joe", or perhaps "Joe Smith," and `coffee` with "a coffee mug." This would result in Halberd translating the planning output `(holding joe coffee)` into the English sentence "Joe Smith is holding a coffee mug". The best part about this solution is that if there was ever a time that Mary, Tom, Angela or any number of other characters could be holding an object of any sort, this same sentence structure could be used and the blanks would simply be filled in with different terms.

While we believed that having the freedom of the formatted sentences described above was very important for authoring a story with a narrative planner, we also recognized the importance of allowing fine-grained and specific details in certain descriptions. We discuss the authorial importance of this specificity in greater detail in the chapter on Authoring Plan-Based Stories for Halberd, under the section titled "Writing Your Story", but I will provide an example here for context. Consider the sentence "John gets married to Jane." This is the type of sentence that our formatted strings could easily produce, and while it is a perfectly valid way of representing the action of getting married, it is exactly the type of story event that could warrant a longer, more dramatic, and more personal description. The fill-in-the-blank structures we offered created a foundation for presenting our stories, but did not offer much in terms of achieving specificity.

In order to add specificity to our options for structuring sentences in Halberd, we built another feature into our engine: keyword-specific sentences. In our JSON file, we provided a method for an author to optionally specify sentences that would be displayed if certain keywords were encountered. A fill-in-the-blank style sentence would still be required in the JSON file, but Halberd would now perform a check for any keyword-specific sentences and look to see if those keywords showed up in the output. For example, if the output was `(marry john jane)` – which could produce the English sentence "John gets married to Jane" – an author may have prepared a sentence for the marriage action that was specific to the keyword `john`: "John has been looking forward to getting married for his entire adult life, and couldn't be happier to have found his dream bride, Jane." Halberd would recognize this keyword and display the specific sentence instead of the basic one.

In addition to formatted strings and keyword-specific sentences, our team added one other function for sentence structure to the Halberd Engine: randomized sentence variants. This means that an author can provide more than one formatted string for describing a single action, and Halberd will choose one of the strings randomly. The need for this became apparent once we started to create and play narratives and found that certain actions would occur many times in a single narrative. For example, in a narrative with multiple locations and multiple characters, characters would often move from place to place. With a single formatted string describing the move action, we would see repetitive sentences such as "Mary walked to the store," "Mark walked to the office," "Bill walked to the park," "Jane walked to the airport," etc. Rather than simply repeating the formatted string "`%1$s` walked to `%2$s`" over and over, our new addition allowed the engine to choose between that string and other variants such as "`%1$s` ran to `%2$s`," "`%1$s` strolled to `%2$s`," "`%1$s` jogged to `%2$s`," etc. This didn't provide any technical purpose, but it allowed us to optionally improve the aesthetics of our game text by diversifying the action descriptions.

One final addition we made to the display options in Halberd was display-types. Any predicate defined in the domain was allowed to be internal or external. If a predicate was internal, the description for that predicate would only be displayed if the predicate was associated with the player. For example, if 'hungry' was an internal predicate, the player would only be informed if the player's character was hungry, but not if one of the other characters was hungry. If a predicate was external, it would be displayed regardless of who it was associated with. Additionally, we introduced a concept of on-stage and off-stage, where an author could choose whether the player would be informed about actions that took place in a different location, or if they had to be in the same area in order to witness it. The authorial benefits of this concept will be discussed in the chapter on Authoring Plan-Based Stories for Halberd, under the section titled "Creating Your World."

2.4 Engine Limitations

Now that I have discussed some of the technical aspects of planning and highlighted the affordances of the Halberd Engine, it is important to understand its limitations.

Halberd is limited by the hardware in the computer it is running on. This can be said for most software, but the content limitations this imposes must be considered when authoring a story using this engine. As previously explained, the Glaive planning algorithm that Halberd uses searches every possible story-path for the fastest path that makes sense to the planner. This set of all possible paths can grow very large, very quickly with the addition of extra characters, items, locations, actions, predicates, and axioms in the PDDL files. This has two important implications for the design process. Firstly, the larger the set of all possible paths becomes, the longer it takes the planning algorithm to examine them all. As Halberd needs to re-plan the narrative after each choice the player makes, this will mean that the player will need to wait through longer loading-times between choices in stories with more elements. Depending on the type of game and the experience that the game developer is trying to create, this can become prohibitive. Secondly, a game that involves a very large number of elements may even produce so many possible story-paths that the software runs out of memory-space on the computer and shuts the game down.

Halberd is also limited by the software. This limitation is less severe, as the code-base can be modified and expanded. However, in its current state, there are considerations that an author should make. While PDDL can define certain logical rules using axioms, there are certain types of rules that a planner is not well-suited to define. For example, if an author wanted to limit the number of objects that a character could pick up, there is no simple way to do this. This is true of many forms of logic that involve keeping track of information such as numbers. This is easy to do in the code that makes up Halberd and Glaive, but there is currently no channel or method for communicating this information between the code and the PDDL files.

PDDL is also not well-suited for generating character dialogue. Some of the sentence structure options that we discussed earlier could be used to embed rudimentary character dialogue into a game, but that dialogue would remain the same every time the associated element needed to be displayed, and it would not be an interactive experience for the player. In its current state, Halberd is best-suited for relaying information as if the story is being told by a narrator. However, it would be possible to extend the engine to use a dialogue generation tool in a future project.

3 Authoring Plan-Based Stories for Halberd

In this section, I will address what it means to author plan-based branching narratives, and discuss the implications of the technical features described in "The Halberd Narrative Planning Engine" in an authorial context. We will also talk about the interactive, iterative process involved with designing and writing games for Halberd. We will suggest some of the best practices for authoring stories with this engine, providing examples from the experience of creating our own story: *The Worm of Everhill*.

3.1 *The Worm of Everhill*

The Worm of Everhill is the first fully-developed game created with the Halberd Narrative Planning Engine. It is a horror-themed narrative that takes place in the small town of Everhill, where a strange worm emerges and begins to control the minds of the unsuspecting townspeople, to violent ends. The player of this game takes control of an unnamed protagonist whose actions can change the course of events in the story. The story itself is written to produce a sense of mystery and fear, as the player is not informed of impending danger and is left to discover suspicious behavior and victims of attacks, and to ultimately reveal the monster behind the violence: the mindworm. Either that, or become a victim themselves.

The Worm of Everhill was developed using techniques that were derived both from research on authoring practices in traditional media, and from the experiences we had while working with Halberd. In the following sections, I will discuss these techniques and refer back to *The Worm of Everhill* to reflect on the effectiveness of their implementations.

3.2 Creating Your World

At the beginning of this chapter, I referred to the authoring process using Halberd as an interactive, iterative process. To elaborate on what that means, I would like to consider a quote from novelist Duana Swierczynski regarding the loose plotting process he used when writing his crime novel, *The Wheelman*:

> [*The Wheelman*] opened with a bank robbery gone wrong and just followed the aftermath. I had a vague idea about where [the novel would] go, but when I sat down to write each section, I allowed the story to be the boss. It was great fun because I was discovering the story as I went along. New characters would pop up and I'd be like, Oh, okay, you want to join in? Sure. So what's your story? The answer would often surprise me (Swierczynski, n.d.).

This process that Swierczynski describes, of sitting in a scene and watching the story play out and welcoming new ideas that present themselves, is exactly the interactivity that I am referring to. While Swierczynski's process of plotting was abstract and internal, it is very similar to the tangible experience that the Halberd Engine offers to the author. Because the engine uses planning software, the narrative does not require being written out before the program is run. This means that an author can set up a world, or even just a scene, and take the role of the player by running the game and experiencing how the story plays out. Our team did this regularly while designing *The Worm of Everhill.* We would create a scenario, run the game, make choices as the player, and watch as the planner responded in turn. In doing this, we got a sense of what felt right, what felt wrong, what was missing, and what did and didn't happen the way we thought it would. This conversation between the author and the game is the interactive element, and using the information we got from this process to tweak and expand our story is the iterative element.

Approaching narrative authorship this way, we found that it's better to start simple. Placing ourselves in a scene to get a sense of the pacing, what we liked and what we didn't, was much easier and less overwhelming when there were fewer elements to consider: fewer characters and items to interact with, and fewer actions available to us. It was easier to feel the absence of something than to interact with a complicated or busy scene and figure out which part was negatively affecting our experience. *The Worm of Everhill* features 9 characters besides the player, 8 locations, 8 items, and nearly 20 actions defined in the PDDL, but until the late stages of its development, the game was tested with smaller-scale scenarios and slowly built-upon using this interactive, iterative process.

In addition to helping determine which PDDL defined elements you should add, remove, or change in your game, interactively experiencing your story with Halberd can also provide valuable insight into the effectiveness of your games goals. There are two types of goals I am referring to here: experience goals, and planner goals. Experience goals are part of the design process for anything an audience will interact with. In a story, your experience goals for the reader may be for them to feel happy or sad, to have learned a lesson, to have experienced something new, or in the case of *The Worm of Everhill*, to feel a sensation such as fear.

As a tradeoff for using planner software to generate a narrative, an author must give up some control over the sequence of events in their story, so it's important to consider experience goals that are not directly tied to events happening in a certain order. In *The Worm of Everhill*, we wanted our story to build tension and suspense, eventually reaching a climax, and ending in a resolution. This is a common flow for traditional literature, but as we could not control what events happened at what times, we took advantage of some of the technical features of the engine, described in our chapter on The Halberd Narrative Planning Engine, and tested and tweaked our game until it produced the feeling we were looking for. We used the concept of on-stage and off-stage to produce mystery. As there were many locations in our game and the player would only be informed of what was happening at their own location, the mindworm would always possess some victim off-stage at the beginning of the story, and there was a high likelihood that the first people to be killed by the mindworm would be killed off-stage, leaving it up to the player to find the bodies and attempt to discover who was responsible.

Because of the nature of the how the planner works, it is perfectly possible for the mindworm or a possessed human to reveal themselves to the player early in the game, or for the player to walk into a scene where a violent character quickly attacks them before they get their bearings. While this doesn't necessarily satisfy the experience goals for our game, we did not consider this a failure. It would have been possible for us to prevent this type of situation from occurring early in our game by authoring additional rules or story goals into our PDDL files, but we found that it did not typically detract from the experience of the game. In fact, when we playtested *The Worm of Everhill*, our testers often considered such an unfortunate event as a learning experience and were excited to restart the story with the new insight that the experience gave them. It is important to consider that one motivation to use a narrative planner for creating games is to allow for a great number of possible story-paths, and some will inevitably be less ideal or less satisfying than others. However, a story ending in a satisfying way is not the same as a story ending in an ideal way.

Janet Burroway, in her book *Writing Fiction: A Guide To Narrative Craft*, explains that a readers satisfaction with a story is not dependent on a happy story ending in a happy way, or a sad story ending sad. Readers are open to being taken on all sorts of adventures through dangers and horrors and disasters alike because they know the story is fiction, and they know it will end (Burroway, 1992, pp. 41–42). She suggests that the plot itself is what one must master to keep the reader turning the page (Burroway, 1992, p. 37). Best-selling author James Patterson's answer to keeping the reader interested is to keep the story suspenseful. Readers want suspense, and Patterson believes that lies in leaving questions that must be answered, and finding the answer to those questions is the driving force that keeps the reader with you (Berkowitz, 2014). It's hard to give too much insight into how this would work in a work in a plan-based narrative, as each story is different and certain questions may be easier to present to the player than others. However, we believe that the mystery created by the off-stage events in *The Worm of Everhill* accomplished this well for a horror setting. Experiencing the story interactively as an author, as well as playtesting it with others, is a helpful way to gain insight into the effectiveness of your story at motivating the player to keep playing.

The second type of goal that we mentioned was a planner goal. As it sounds, this is the goal that the planner itself is planning toward. When described in the chapter on The Halberd Narrative Planning Engine under the section "How Glaive Makes Choices," planner goals are treated as the authorial goal. For example, a PDDL could set a goal for the planner to state that, at the end of the story, John and Jane needed to be married. However, if the player was in control of one of these two characters and also working toward that goal, the story may not be very interesting. As a planner tries to find the quickest path to its goal, having the same goal as the player causes everything to work in favor of the player's actions, guiding them quickly and easily to this planner goal. That could be fine, if that outcome matches the experience that the author wanted the player to have, but it lacks what Burroway calls the "first encountered and fundamental element of fiction": conflict. According to Burroway, "in literature, only trouble is interesting" (Burroway, 1992, p. 39). I believe that this can be applied to narratives in games as well, so while authoring *The Worm of Everhill*, steps were taken to ensure that conflict was authored into the story. The following section will discuss how this can be done, and how it can be enhanced by using Glaive's ability to reason about character intentionality.

3.3 Conceptual Challenge: Conflict and Intentionality

At the end of the last section, we briefly mentioned writing goals for the planner to plan toward, and how writing planner goals that are in-sync with player goals, while perfectly valid, may not produce satisfying story results due to the lack of an essential element: conflict. Conflict is a highly important element in fiction as it often drives character development and serves as motivation for the reader to keep reading. Conflict is also important in digital games, as similar rules for a satisfying narrative apply to games, and the audience is predisposed to expect conflict as a formative element of gameplay. Using Halberd, there are a several ways for an author to introduce conflict into their game: planning against the player, and writing intentional characters.

While designing a story for Halberd, it may be an author's first instinct to set the planner's goal to whatever the goal of player is. For example, in *The Worm of Everhill*, the town and townspeople would be in danger during the story and we expected that the player would want to save them, so our initial thought was to have Halberd plan toward a solution where the town and its people were safe. Halberd could easily make a plan to get to this solution, but there was a problem: none of the townspeople were ever put in danger. Halberd wouldn't create the horrific scenario that we wanted our players to experience, because putting the townspeople in danger did nothing to accomplish the goal of the planner, which was to make everyone safe.

The solution to this problem that we found was to author the goal of the planner as a conflict against the player. Since we expected that the player would want to save the town and the people, we instead changed the goal of the planner to find some solution where everyone in the town had been killed off, and the mindworm was victorious. Suddenly, our story became interesting; the player was put into a scenario where the mindworm was possessing townspeople and using them to kill other townspeople in an attempt to satisfy the planner's goal of having everyone dead. Now the player had something to battle against. The story became a race against the clock to discover the murders, reveal the monster, and get rid of it before it destroyed the entire town or killed the player character directly. This one change to the goal of our story made all of the difference.

Sometimes an author may not want the entire game world to plot against the player, however, and Halberd provides a way to manage this as well. In addition to writing planner goals, an author using Halberd can write character-specific goals by using the feature of character intentionality. The Glaive Narrative Planner, which Halberd extends, contains logic to reason about a character's intentionality before they take any actions. What this means is that the author can specify some intention or goal for a character – for example, the mindworm intends to kill all of the townspeople – and will only make a character take an action that makes sense in terms of that intention or goal. This is a necessary feature to preserve the believability of a character, as a planner trying to find the fastest solution to kill everyone in the town might simply have each of the characters find a weapon and kill themselves, rather than having the mindworm travel from place to place and do all of the work. Luckily, as this solution does not make sense with the character goals of the non-possessed characters, the planner will instead find the fastest plan that also satisfies a character's goals: in this case having the mindworm kill all of the townspeople.

Character intentionality can also be used as a tool to create conflict in a story. If two characters have conflicting intentions, it is likely that a conflict will occur over those intentions at some point in the story. A character can also be written to have intentions that conflict with the expected player's intentions. This can make the game's conflict feel more personal, and also avoid the issue of the entire world working against the player that can occur when the planner goal contradicts the player's goal.

3.4 Conceptual Challenge: Rigidity

Combinations of planner goals and character intentionality, as discussed in the previous section, can greatly increase an author's control over the player's experience. However, as we increase our knowledge of the ways in which we can control the story, we should take a moment to recall the purpose of authoring these branching narratives: to creative an interactive experience where a story can be generated in many different ways to adapt to the player's choices. While taking advantage of these tools for controlling the player's experience, an author should give some careful thought to the rigidity of their story.

Let's talk about what rigidity means in the context of an interactive game. This comes back to the issue of the narrative paradox described in the Introduction: the more control an author asserts over a game's narrative, the less interactive it becomes. While Halberd produces branching narratives to address this problem, it is possible for an author using Halberd to force various degrees of control over the course of the events in-game. For example, a game may be able to branch many times, but the author can enforce heavy control by setting the only goal of the story to end in a specific room that the player needs a key to get into. This means that no matter what other actions the player takes along the course of the narrative, they will always need to get that key and enter that room in that specific order, essentially folding all possible branches of the story back in to a certain point. This type of control can be helpful, but it can also be frustrating for a player, making them feel like they were simply given the illusion of choice.

I will offer another example of this rigidity problem through a story that I wrote during my preliminary experiments with PDDL that, in my opinion, failed to properly address this challenge. The story was a simple spy narrative, drawing inspiration from the gadget using agents in movies such as James Bond. In the story, a spy had to infiltrate a building by sneaking past a guard, gain access to the mastermind's office, obtain a key code to a vault accessible from the lobby, enter the vault, steal the secret plans, and finally escape. While simple and not terribly exciting, the structure of the story was fine, but the problem lied in the fact that it really only ever played out in one of two ways: either the spy would succeed in completing each of the objectives listed above, or he would be caught and killed. While writing the story, I considered the series of events that may occur during the plan and tried to control the progression too heavily by using that lock-and-key structure discussed above. In this story, you had to sneak past the guard to get to the mastermind's office, you had to obtain the key code before you could get into the vault, and you had to have the secret plans before you could escape. The player may be able to mess around and try a couple of different things, but ultimately they would need to do these specific actions in this specific order to finish the story without the spy dying.

I would like to note, however, that rigidity is not something that needs to be avoided entirely. In fact, it is difficult to avoid using some amount of this control and still author a satisfying story. A narrative should be more rigid in places that are more import to the story and the experience of the player. For example, if an author felt that two characters needed to get married before the end of the story, they could include this as part of the goal and Halberd would work to enforce it. If the author left many elements of this marriage up to the planner, giving many options for proposal locations, types of rings, wedding theme, which two characters were getting married, etc., the occurrence of the event would be rigid, but the details would not. In *The Worm of Everhill*, we enforce some rigidity by making a character need a weapon before they can target anyone, and making them need to target someone before they can kill that person. These extra steps can also help slow the pacing of the story a bit, balancing the speed of a human player who might want to explore the world, and the speed of the planner that is simply taking steps to reach its goal. Determining the best balance of rigidity should be another part of the interactive, iterative experience of working with Halberd.

3.5 Writing Your Story

In the sections above, we discussed some of the best authorial practices for taking advantage of the planning features offered by Halberd. We talked about developing the world, bit by bit, plotting conflict and character intentionality, and how to properly assert control over your narrative without sacrificing the player's agency. What we have not yet discussed is how to use the formatted strings, described in the section on Displaying Text to the Player, to actually write the sentences that will make up the story that is presented to the player. This section will not cover how to write a good sentence, but instead focus on how to use the PDDL to English-sentence translation tools that Halberd has to offer, and how the different ways of authoring these sentences can affect the story.

As a reminder, Halberd offers three types of sentence definitions: fill-in-the-blank, keyword-specific, and random variants. The structure of one of these sentences can be a powerful tool for affecting the player's experience. Stanley Fish, in his book *How to Write a Sentence and How to Read One*, claims that much of the power in a sentence comes from its syntax; "It is syntax that gives the words the power to relate to each other in a sequence ... to carry meaning — of whatever kind — as well as glow individually in just the right place" (Fish, 2011, p. 8). "Joe jogged to the house" has a very different feeling than "It was to the house that Joe jogged," and in our engine this is simply a matter of rearranging the basic sentence structure and putting the keywords in different places.

The fill-in-the-blank style sentences are likely to be the most commonly used structure in your generated narrative. These structures use one or more 'blanks' to generalize the description of a predicate or an action and allow the planner to fill in those blanks as necessary. The pick-up action in a story could, for example, be represented by the structure "______ picks up ______," which would allow for the translation of any situation in which any character picked up any object. This is a very simplistic example, but an author using this structure can write whatever they want before, after, and between these blanks, or even rearrange the order of the blanks themselves. This is why we can transform a sentence like "Joe jogged to the house" into "It was to the house that Joe jogged." It is also not required that these structures are kept to a single sentence. An author could, if they desired, write paragraphs to describe these actions. However, when elaborating heavily on an action in the description, it can become difficult to allow the sentence to remain general to any character, place, item, etc.

Sometimes generality does not meet the needs of an author, and a sentence may require elaboration that would not be appropriate for anything simply filling in the blanks. Janet Burroway, in her book *Writing Fiction: A Guide To Narrative Craft*, emphasizes the importance of specificity. Burroway claims that an author, while thinking in terms of the basic categories of a story and conflict, may get lost in the idea that the events of a story take place in an abstract dimension. Man against nature is one such abstract dimension, and is one of the basic conflicts found in a story, but it doesn't truly become interesting until man and nature are defined and we are given the specifics of the situation to empathize with (Burroway, 1992, p. 41). This concept is highly relevant to the nature of the plan-based narratives we were creating. The fill-in-the-blank structures Halberd offers create a foundation for presenting our stories, but do not offer much in terms of specificity. For example, if there was an action to hack a computer, rather than plainly stating "Mary hacked the computer," an author might want to say something like "Mary was a local legend when it came to hacking into computer systems, and she gained full access to the network within seconds." However, if the only blank in this sentence was for the character, we could run into undesirable situations in the narrative. For example, if Henry were to be the one hacking, we might not want to say "Henry was a local legend when it came to hacking into computer systems, and she gained full access to the network within seconds." First off, the pronoun 'she' may not be appropriate for Henry, and secondly we probably don't want every single character who hacks a computer in the narrative to be a local legend.

To address this, we extended the use of the generalized formatted strings, offering an optional feature to recognize a keyword that is filling in one of the blanks. For example, an author could write a specific sentence that the program would choose to display if it recognized the keyword "Mary," but if the program didn't find that keyword, it would default to the generic fill-in-the-blank method. For example "Henry hacked the computer." These optional, keyword-specific sentences should be used sparingly or one may find themselves again running into the narrative paradox, writing a sentence for every possible combination of characters, actions and objects. Used responsibly, the allowance of this kind of control gives the author the ability to insert more detail and perhaps create deeper points of storytelling.

It is important to note that, while an author may have the time to write a long and specific sentence about every action in their story, it may not always be beneficial to do so. In our plan-based system, the author has to give up some control over the order of the events in a narrative and, as such, cannot know exactly what will be happening when the player is presented with the information about any given world-state. In an interview, author James Patterson offers an anecdote about writing the way that he would verbally tell a story:

> I think what hooks people into my stories is the pace. I try to leave out the parts people skip. I used to live across the street from Alexander Haig, and if I told you a story that I went out to get the paper and Haig was laying in the driveway, and then I went on for 20 minutes describing the architecture on the street and the way the palm trees were, you'd feel like "Stop with the description–what's going on with Haig?"

Halberd will, in fact, describe everything that is in an area to the player if there is a description available. By utilizing the presentation methods we have discussed, an author has some control over this. Typically, the short, cookie-cutter sentences like “a chair is in the living room” will not take too much time to read and can be easily ignored if there is something more interesting that the player wants to pay attention to. However, if an author decides to write a paragraph to describe that chair, focusing on all of the little scuffs on the pattern or the pocket-change between the seat cushions, it will be harder for the player to move past that chair and onto a more interesting focus. Now, consider that the chair may just be one of ten items in a room that are each described in such excruciating detail; the story would end up resembling Patterson’s twenty-minute tangent about his neighborhood. To avoid this, the author of a plan-based story should keep most descriptions of items and events short, elaborating only on points of drama or interest: events that are critical to the story that the author wants to tell, or that the author would like to emphasize. In many stories, the action of one character killing another character would be a greater point of interest than a character making a sandwich.

4 Conclusion

I would like to take a moment to reflect on The Halberd Narrative Planning Engine. The goal of this project was to address the narrative paradox and to present a solution to the large branching problem in interactive narratives. In many ways, I feel that Halberd accomplished this. The engine allows for an author to define a world, rules, and goals, and handles the planning of the many story-branches itself. The design process of these authored worlds does require the author to work within constraints that are not found in traditional media, but the trade-off in labor can be well worth it as the planner will be able to quickly generate paths that would take considerable effort to manually write. Additionally, while the author has to give up some control over the sequence of events in a story, Halberds tools for asserting control and imparting conflict into the story as part of its planning process, successfully blend player agency and story events in a way that is not seen in other forms of storytelling in games.

Halberd's greatest weaknesses come from its dependency on the computer that it is running on. The high complexity of the planning algorithm used in Halberd means that complex stories take a long time to re-plan between the player's actions, and this can quickly become prohibitively long. In addition, depending on the hardware, there is a complexity threshold past which Halberd will run out of memory and stop the program from running. Luckily, these limitations will be minimized as technology continues to improve over the years to come, and plan-based story generation will become much more feasible for complex narratives. Because of this, planning software may play a large role not only in the design of games in the future, but in the authorship of interactive storytelling as a whole.

Halberd extends the narrative planning work found in Stephen Ware's Glaive by taking the concept of planning into a dynamic context, helping an author to incorporate player agency directly into the events of a story by procedurally adapting the narrative to the player's actions. This context provides a new avenue for storytelling, both for game developers and authors, and takes a step toward solving the narrative paradox that plagues interactive storytelling. Planning-based games could potentially create a whole new level of audience immersion, allowing players to have their actions truly impact the digital world. As more advanced technology expands their capabilities, I am excited to see where engines like Halberd will take the field of interactive media.

References

Aylett, R. (2000). Emergent narrative, social immersion and "storification". In *Proceedings of the 1st international workshop on narrative and interactive learning environments* (pp. 35–44).

Barber, H. & Kudenko, D. (2007). Dynamic generation of dilemma-based interactive narratives. *AIIDE*, *7*, 2–7.

Bell, I. (n.d.). Ian bell's elite pages. Retrieved April 10, 2017, from http://www.iancgbell.clara.net/elite/

Berkowitz, J. (2014, April). World's best-selling author james patterson on how to write an unputdownable story. Retrieved April 1, 2017, from https://www.fastcompany.com/3029052/worlds-best-selling-author-james-patterson-on-how-to-write-an-unputdownable-story

Burroway, J. (1992). *Writing fiction: a guide to narrative craft* (3rd). HarperCollins.

Dehn, N. (1981). Story generation after TALE-SPIN. In *Ijcai* (Vol. 81, pp. 16–18).

Fikes, R. E. & Nilsson, N. J. (1971). STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial intelligence*, *2*(3-4), 189–208.

Fish, S. (2011). *How to write a sentence and how to read one.* HarperCollins.

Hoffmann, J. & Nebel, B. (2001). The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Karlsson, B., Ciarlini, A., Feijó, B., & Furtado, A. (2006). Applying a plan-recognition / plan-generation paradigm to interactive storytelling: the LOGTELL case study. *Monografias em Ciência da Computação Series.*

Mateas, M. & Stern, A. (2003). Façade: an experiment in building a fully-realized interactive drama. In *Game developers conference* (Vol. 2).

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., ... Wilkins, D. (1998). PDDL - the planning domain definition language.

Meehan, J. R. (1977). TALE-SPIN, an interactive program that writes stories. In *IJCAI* (Vol. 77, pp. 91–98).

Montfort, N. (2005). *Twisty little passages: an approach to interactive fiction.* MIT Press.

The Epic of Gilgamesh: The First Epic, from The First Civilization. (n.d.). Retrieved March 17, 2017, from http://webpages.uidaho.edu/engl257/Ancient/epic_of_gilgamesh.htm

Riedl, M. O. & Young, R. M. (2010). Narrative planning: balancing plot and character. *Journal of Artificial Intelligence Research.*

Swierczynski, D. (n.d.). Tumbling down a hill in a house that is on fire. Retrieved from http://www.centerforfiction.org/forwriters/writers-on-writing/tumbling-down-a-hill-in-a-house-that-is-on-fire/

Togelius, J., Shaker, N., & Nelson, M. J. (2016). Introduction. In N. Shaker, J. Togelius, & M. J. Nelson (Eds.), *Procedural content generation in games: a textbook and an overview of current research* (pp. 1–15). Springer.

Toy, M. C. & Arnold, K. C. (2012). A guide to the Dungeons of Doom. *Computer Systems Research Group. University of California, Berkeley. Nd Web*, *9*.

Ware, S. G. & Young, R. M. (2011). CPOCL: a narrative planner supporting conflict. In *Seventh artificial intelligence and interactive digital entertainment conference*.

Ware, S. G. & Young, R. M. (2014). Glaive: a state-space narrative planner supporting intentionality and conflict. In *Aiide*.

Young, R. M. (1999). Notes on the use of plan structures in the creation of interactive plot. In *AAAI fall symposium on narrative intelligence* (pp. 164–167).

Young, R. M., Riedl, M. O., Branly, M., Jhala, A., Martin, R., & Saretto, C. (2004). An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*, *1*(1), 51–70.

Young, R. M., Ware, S. G., Cassell, B. A., & Robertson, J. (2013). Plans and planning in narrative generation: a review of plan-based approaches to the generation of story, discourse and interactivity in narratives. *Sprache und Datenverarbeitung, Special Issue on Formal and Computational Models of Narrative*, *37*(1-2), 41–64.

Appendices

A PDDL Domain File for *The Worm of Everhill*

```
(define (domain horror)
  (:requirements :adl :domain-axioms :intentionality)
  (:types entity immovable - object
          actor item - entity
          human mindworm - actor
          player npc - human
          place meta - immovable
          weapon tool combustible restraint igniter - item
          number)
  (:predicates (dead ?entity - entity)
               (at ?entity - entity ?place - place)
               (armed ?human - human)
               (has ?human - human ?item - item)
               (controlling ?mindworm - mindworm ?human -
                  human)
               (possessing ?mindworm - mindworm)
               (possessed ?actor - actor)
               (suspicious ?human - human)
               (doused ?actor - actor)
               (scorched ?actor - actor)
               (targeting ?attacker - human ?attacked -
                  human)
               (goal-achieved ?meta - meta)
               (destroyed-town ?mindworm - mindworm)
               (saved-town ?player - player)
               (free ?mindworm - mindworm)
               (bound ?human)
               (emerged ?mindworm - mindworm))

  ;;----------ACTIONS----------

  ;; A human moves from one place to an adjacent place.
  (:action move
    :parameters   (?actor - actor ?from - place ?to -
       place)
    :precondition (and (not (= ?from ?to))
                       (not (dead ?actor))
```

```
                     (at ?actor ?from))
  :effect       (and (not (at ?actor ?from))
                     (at ?actor ?to))
  :agents       (?actor))

;; An attacking human targets a victim for attack
(:action target-human
  :parameters   (?attacker - human ?victim - human ?
     weapon - weapon ?place - place)
  :precondition (and (not (dead ?attacker))
                     (not (dead ?victim))
                     (at ?attacker ?place)
                     (at ?victim ?place)
                     (has ?attacker ?weapon)
                     (not (= ?attacker ?victim)))
  :effect       (targeting ?attacker ?victim)
  :agents       (?attacker))


;; An armed human kills another human that is in the
   same area
(:action kill
  :parameters   (?killer - human ?victim - human ?place
     - place ?weapon - weapon)
  :precondition (and (not (dead ?killer))
                     (not (dead ?victim))
                     (targeting ?killer ?victim)
                     (not (= ?killer ?victim))
                     (at ?killer ?place)
                     (at ?victim ?place)
                     (not (possessed ?victim))
                     (has ?killer ?weapon))
  :effect       (and (dead ?victim)
                     (not (targeting ?killer ?victim)))
  :agents       (?killer))

;; An armed human kills a possessed npc that is in the
   same area
(:action kill-possessed-human
  :parameters   (?killer - human ?victim - npc ?mindworm
      - mindworm ?place - place ?weapon - weapon)
  :precondition (and (not (dead ?killer))
                      (not (dead ?victim))
                      (targeting ?killer ?victim)
                      (controlling ?mindworm ?victim)
                      (not (= ?killer ?victim))
                      (at ?killer ?place)
                      (at ?victim ?place)
                      (has ?killer ?weapon))
  :effect        (and (dead ?victim)
                 (not (targeting ?killer ?victim))
                 (not (free ?mindworm))
                 (intends ?mindworm (free ?mindworm)))
  :agents        (?killer))


;; An armed human kills themself
(:action kill-self
  :parameters    (?npc - npc ?weapon - weapon)
  :precondition (and (not (dead ?npc))
                      (has ?npc ?weapon))
  :effect        (dead ?npc)
  :agents        (?npc))


;; A possessed human takes an item from a dead victim.
(:action loot-body
  :parameters    (?taker - human ?victim - human ?item -
     item ?place - place)
  :precondition (and (not (dead ?taker))
                      (not (= ?taker ?victim))
                      (dead ?victim)
                      (at ?victim ?place)
                      (at ?taker ?place)
                      (possessed ?taker)
                      (has ?victim ?item))
  :effect        (and (not (has ?victim ?item))
                      (has ?taker ?item))
  :agents        (?taker))


;; The player takes an item from a dead victim.
(:action player-loot-body
  :parameters    (?taker - player ?victim - human ?item -
      item ?place - place)
  :precondition (and (not (dead ?taker))
                     (not (= ?taker ?victim))
                     (dead ?victim)
                     (at ?victim ?place)
                     (at ?taker ?place)
                     (has ?victim ?item))
  :effect       (and (not (has ?victim ?item))
                     (has ?taker ?item))
  :agents       (?taker))


;; The player picks up an item located at the place they
   're currently at
(:action player-pickup
  :parameters   (?taker - player ?item - item ?place -
    place)
  :precondition (and (not (has ?taker ?item))
                     (not (dead ?taker))
                     (at ?item ?place)
                     (at ?taker ?place))
  :effect       (and (has ?taker ?item)
                     (not (at ?item ?place)))
  :agents       (?taker))

;; A possessed human picks up an item located at the
   place they're currently at
(:action pickup
  :parameters   (?taker - human ?item - item ?place -
    place)
  :precondition (and (not (has ?taker ?item))
                     (not (dead ?taker))
                     (possessed ?taker)
                     (at ?item ?place)
                     (at ?taker ?place))
  :effect       (and (has ?taker ?item)
                     (not (at ?item ?place)))
  :agents       (?taker))


;; A human puts down an item at the place they're
   currently at
```

```
(:action putdown
  :parameters   (?dropper - human ?item - item ?place -
     place)
  :precondition (and (has ?dropper ?item)
                     (not (dead ?dropper))
                     (at ?dropper ?place))
  :effect       (and (not (has ?dropper ?item))
                     (at ?item ?place))
  :agents       (?dropper))


;; A human douses a target in a combustible substance
(:action douse-person
  :parameters   (?douser - human ?victim - actor ?
     combustible - combustible ?place - place)
  :precondition (and (has ?douser ?combustible)
                     (not (dead ?douser))
                     (at ?douser ?place)
                     (at ?victim ?place))
  :effect       (doused ?victim)
  :agents       (?douser))


;; A human with an igniter ignites a doused target
(:action ignite
  :parameters   (?killer - human ?victim - actor ?place
     - place ?igniter - igniter)
  :precondition (and (not (dead ?killer))
                     (at ?killer ?place)
                     (at ?victim ?place)
                     (doused ?victim)
                     (has ?killer ?igniter)
                     (not (possessed ?victim)))
  :effect       (and (dead ?victim)
                     (scorched ?victim))
  :agents       (?killer))


;; A human with an igniter ignites a doused possessed
   target
(:action ignite-possessed
  :parameters   (?killer - human ?victim - human ?
     mindworm - mindworm ?place - place ?igniter -
     igniter)
  :precondition (and (not (dead ?killer))
                     (at ?killer ?place)
                     (at ?victim ?place)
                     (controlling ?mindworm ?victim)
                     (doused ?victim)
                     (has ?killer ?igniter))
  :effect       (and (dead ?victim)
                     (scorched ?victim)
                     (dead ?mindworm))
  :agents       (?killer))


;; A mindworm possesses a non-player-human
(:action possess-human
  :parameters   (?mindworm - mindworm ?human - npc ?
     place - place)
  :precondition (and (not (dead ?mindworm))
                     (not (dead ?human))
                     (not (bound ?human))
                     (not (possessed ?human))
                     (not (possessing ?mindworm))
                     (at ?mindworm ?place)
                     (at ?human ?place))
  :effect       (and (not (at ?mindworm ?place))
                     (controlling ?mindworm ?human)
                     (possessed ?human)
                     (possessing ?mindworm)
                     (intends ?human (destroyed-town ?
                        mindworm))
                     (intends ?human (armed ?human)))
  :agents       (?mindworm))


;; A mindworm leaves a possessed non-player-human
(:action leave-possessed-human
  :parameters   (?mindworm - mindworm ?human - npc ?
     place - place)
  :precondition (and (not (dead ?mindworm))
                     (possessed ?human)
                     (controlling ?mindworm ?human)
```

```
                      (at ?human ?place))
  :effect       (and (not (controlling ?mindworm ?human)
    )
                      (not (possessed ?human))
                      (not (possessing ?mindworm))
                      (at ?mindworm ?place)
                      (free ?mindworm)
                      (not (intends ?human (destroyed-
                         town ?mindworm)))
                      (not (intends ?human (armed ?human)
                         ))
                      (intends ?mindworm (possessing ?
                         mindworm)))
  :agents       (?mindworm))


;; The player talks to a non-possessed human
(:action talk-to-human
  :parameters   (?talker - player ?listener - human ?
    place - place)
  :precondition (and (at ?talker ?place)
                      (at ?listener ?place)
                      (not (= ?talker ?listener))
                      (not (dead ?talker))
                      (not (dead ?listener))
                      (not (possessed ?listener)))
  :effect       (not (suspicious ?listener))
  :agents       (?talker))

;; The player talks to a possessed human
(:action talk-to-possessed-human
  :parameters   (?talker - player ?listener - human ?
    place - place)
  :precondition (and (at ?talker ?place)
                      (at ?listener ?place)
                      (not (= ?talker ?listener))
                      (not (dead ?talker))
                      (not (dead ?listener))
                      (possessed ?listener))
  :effect       (suspicious ?listener)
  :agents       (?talker))
```

```
;; A human binds an npc using a restraint item
(:action bind
  :parameters   (?binder - human ?victim - npc ?
     restraint - restraint ?place - place)
  :precondition (and (at ?binder ?place)
                     (at ?victim ?place)
                     (not (dead ?binder))
                     (not (dead ?victim))
                     (not (bound ?victim))
                     (not (possessed ?victim))
                     (has ?binder ?restraint)
                     (not (= ?binder ?victim)))
  :effect       (and (not (has ?binder ?restraint))
                     (bound ?victim))
  :agents       (?binder))

;; A human binds a possessed npc using a restraint item
(:action bind-possessed-human
  :parameters   (?binder - human ?victim - npc ?mindworm
      - mindworm ?restraint - restraint ?place - place)
 :precondition (and (at ?binder ?place)
                     (at ?victim ?place)
                     (not (dead ?binder))
                     (not (dead ?victim))
                     (not (bound ?victim))
                     (controlling ?mindworm ?victim)
                     (has ?binder ?restraint)
                     (not (= ?binder ?victim)))
  :effect       (and (not (has ?binder ?restraint))
                     (bound ?victim)
                     (not (controlling ?mindworm ?victim
                        ))
                     (not (possessed ?victim))
                     (not (possessing ?mindworm))
                     (at ?mindworm ?place)
                     (free ?mindworm)
                     (not (intends ?victim (destroyed-
                        town ?mindworm)))
                     (not (intends ?victim (armed ?
                        victim)))
                     (intends ?mindworm (possessing ?
                        mindworm)))
  :agents       (?binder))


;; A mindworm emerges...
(:action emerge
  :parameters   (?mindworm - mindworm ?place - place)
  :precondition (not (emerged ?mindworm))
  :effect       (and (emerged ?mindworm)
                     (at ?mindworm ?place))
  :agents       (?mindworm))


;;----------AXIOMS----------

;; When a human has a weapon, they are armed
(:axiom
  :vars    (?human - human)
  :context (and (not (armed ?human))
                (exists (?w - weapon)
                        (has ?human ?w)))
  :implies (armed ?human))


;; When a human has no weapon, they are not armed
(:axiom
  :vars    (?human - human)
  :context (and (armed ?human)
                (forall (?w - weapon)
                        (not (has ?human ?w))))
  :implies (not (armed ?human)))


;; When an attackers target is not in the same place as
   the attacker, the attacker stops targeting them
(:axiom
  :vars    (?attacker - npc ?target - npc ?place - place
     )
  :context (and (targeting ?attacker ?target)
                (at ?attacker ?place)
                (not (at ?target ?place)))
  :implies (not (targeting ?attacker ?target)))
  (:axiom
    :vars    (?mindworm - mindworm)
    :context (forall (?h - human)
                     (dead ?h))
    :implies (destroyed-town ?mindworm))


  (:axiom
    :vars    (?player - player)
    :context (forall (?m - mindworm)
                     (dead ?m))
    :implies (saved-town ?player))


  (:axiom
    :vars    (?meta - meta ?mindworm - mindworm)
    :context (and (destroyed-town ?mindworm)
                  (free ?mindworm))
    :implies (goal-achieved ?meta))


  (axiom
    :vars    (?meta - meta ?player - player)
    :context (saved-town ?player)
    :implies (goal-achieved ?meta))
)
```

B PDDL Problem File for *The Worm of Everhill*

```
(define (problem horror)
  (:domain horror)
  (:objects player - player
            meta - meta
            groundskeeper sheriff mayor farmer citizen-one
                citizen-two doctor storeowner - npc
            mindworm - mindworm
            general-store town-hall home sheriff-office
               house-one house-two hospital farm - place
            knife gun pitchfork - weapon
            gasoline alcohol - combustible
            handcuffs duct-tape - restraint
            food - consumable
            lighter - igniter)
  (:init
           (at sheriff sheriff-office)
           (at player home)
           (at groundskeeper hospital)
           (at mayor town-hall)
           (at farmer farm)
           (at citizen-one house-one)
           (at citizen-two house-two)
           (at doctor hospital)
           (at storeowner general-store)
           (at knife house-one)
           (at pitchfork farm)
           (at gun home)
           (at gasoline general-store)
           (at handcuffs sheriff-office)
           (at duct-tape house-one)
           (at lighter house-one)
           (at alcohol house-two)
           (free mindworm)
           (intends mindworm (possessing mindworm))
           (intends meta (goal-achieved meta)))
  (:goal
           (goal-achieved meta)))
```

C Text Decoration File for *The Worm of Everhill*

```
{
  "actions": [
    {
      "name": "wait",
      "description": "%1$s waits",
      "command": "wait",
      "type": "other"
    },
    {
      "name": "move",
      "description": [
        "%1$s moves to %3$s",
        "%1$s travels to %3$s",
        "%1$s jogs to %3$s",
        "%1$s walks to %3$s",
        "%1$s runs to %3$s"
      ],
      "command": "move to %3$s",
      "type": "move",
      "variant-argument" : 1,
      "variants" : [
        {
          "key" : "mindworm",
          "description" : "%1$s quickly crawls off towards
              %3$s",
          "command" : "move to %3$s"
        }
      ]
    },
    {
      "name": "target-human",
      "description": "%1$s approaches %2$s menacingly",
      "command": "approach %2$s menacingly",
      "type": "other",
      "variant-argument" : 3,
      "variants" : [
        {
          "key" : "gun",
          "description" : "%1$s lines up a shot towards %2
             $s",
          "command" : "aim at %2$s"
        },
        {
          "key" : "pitchfork",
          "description" : "%1$s approaches %2$s menacingly
             , pitchfork held high",
          "command" : "approach %2$s, pitchfork raised"
        }
      ]
    },
    {
      "name": "kill",
      "description": "%1$s kills %2$s with %4$s",
      "command": "kill %2$s with %4$s",
      "type": "other"
    },
    {
      "name": "kill-possessed-human",
      "description": "%1$s kills %2$s with %5$s",
      "command": "kill %2$s with %5$s",
      "type": "other"
    },
    {
      "name": "kill-self",
      "description": "%1$s commits suicide with %2$s",
      "command": "commit suicide with %2$s",
      "type": "other"
    },
    {
      "name": "loot-body",
      "description": "%1$s loots %3$s from %2$s's body",
      "command": "loot %3$s from %2$s's body",
      "type": "other"
    },
    {
      "name": "player-loot-body",
      "description": "%1$s loots %3$s from %2$s's body",
      "command": "loot %3$s from %2$s's body",
      "type": "other"
    },
    {
      "name": "player-pickup",
      "description": "%1$s picks up %2$s",
      "command": "pick up %2$s",
      "type": "other"
    },
    {
      "name": "pickup",
      "description": "%1$s picks up %2$s",
      "command": "pick up %2$s",
      "type": "other"
    },
    {
      "name": "putdown",
      "description": "%1$s puts down %2$s",
      "command": "put down %2$s",
      "type": "other"
    },
    {
      "name": "douse-person",
      "description": "%1$s douses %2$s in %3$s",
      "command": "douse %2$s with %3$s",
      "type": "other"
    },
    {
      "name": "ignite",
      "description": "%1$s lights %2$s on fire with %4$s",
      "command": "light %2$s on fire with %4$s",
      "type": "other"
    },
    {
      "name": "ignite-possessed",
      "description": "%1$s lights %2$s on fire with %5$s",
      "command": "light %2$s on fire with %5$s",
      "type": "other"
    },
    {
      "name": "possess-human",
      "description": "%1$s enters %2$s through the ear
         canal",
      "command": "enter %2$s through the ear canal",
      "type": "other"
    },
    {
      "name": "leave-possessed-human",
      "description": "%1$s erupts from %2$s's nostril",
      "command": "erupt from %2$s nostril",
      "type": "other"
    },
    {
      "name": "talk-to-human",
      "description": "%1$s has a pleasant conversation
         with %2$s",
      "command": "Talk to %2$s",
      "type": "other"
    },
    {
      "name": "talk-to-possessed-human",
      "description": "%1$s tries to start a conversation
         with %2$s, but %2$s remains eerily silent",
      "command": "Talk to %2$s",
      "type": "other"
    },
    {
      "name": "bind",
      "description": "%1$s restrains %2$s using the %3$s",
      "command": "Restrain %2$s with %3$s",
      "type": "other"
    },
    {
      "name": "bind-possessed-human",
      "description": "%1$s restrains %2$s using the %4$s,
         and a worm erupts from %2$s's nostril",
      "command": "Restrain %2$s with %4$s",
      "type": "other"
    },
    {
      "name": "emerge",
      "description": "%1$s emerges into %2$s",
      "command": "Emerge",
      "type": "other"
    }
  ],
  "predicates": [
    {
      "name": "dead",
"description": "%1$s is dead",
"type": "external",
"variant-argument" : 1,
"variants" : [
  {
    "key" : "player",
    "description" : "You are left laying in a pool
       of your own blood. Flies begin to surround
       your fresh wounds as a metallic odor tinges
       the air, and your body lies cold and
       motionless"
  },
  {
    "key" : "groundskeeper",
    "description" : "%1$s lies lifeless on the
       ground. She looks sad but strangely peaceful,
        her colorful dress drenched in blood"
  },
  {
    "key" : "mayor",
    "description" : "%1$s's body lays in a heap. A
       campaign pin, \"Ableton for mayor\", lies at
       his side, stained with blood"
  },
  {
    "key" : "farmer",
    "description" : "%1$s is dead, but his eyes
       remain open. His hand is twisted, almost
       strained, as if grasping for something"
  },
  {
    "key" : "citizen-one",
    "description" : "%1$s is dead, one hand
       clutching his chest, the other reaching out
       for something or someone. He was not ready to
        go"
  },
  {
    "key" : "citizen-two",
    "description" : "%1$s's body is soaked in blood,
        laying still and breathless on the ground.
       She had put up a fight"
    },
    {
      "key" : "doctor",
      "description" : "%1$s lays dead on the ground,
         blood pooling around him. His teeth are bared
          in a grimace"
    },
    {
      "key" : "storeowner",
      "description" : "%1$s's body lays quiet. She is
         dead. She was the oldest person in town, but
         she couldn't survive this day"
    }
  ]
},
{
  "name": "at",
  "description": [
    "%1$s is at %2$s"
  ],
  "type": "locational"
},
{
  "name": "armed",
  "description": "%1$s is armed",
  "type": "external"
},
{
  "name": "has",
  "description": "%1$s has %2$s",
  "type": "external"
},
{
  "name": "controlling",
  "description": "%1$s is controlling %2$s",
  "type": "internal"
},
{
  "name": "possessed",
  "description": "%1$s is possessed",
  "type": "internal"
},
{
  "name": "possessing",
  "description": "%1$s is possessing a human",
  "type": "internal"
},
{
  "name": "suspicious",
  "description": "%1$s is acting suspiciously",
  "type": "external"
},
{
  "name": "doused",
  "description": "%1$s is doused in a flammable liquid
    ",
  "type": "external"
},
{
  "name": "scorched",
  "description": "%1$s's body is burned to a crisp",
  "type": "external"
},
{
  "name": "targeting",
  "description": "%1$s is focused on %2$s",
  "type": "internal"
},
{
  "name": "goal-achieved",
  "description": "The story is complete",
  "type": "external"
},
{
  "name": "destroyed-town",
  "description": "Everyone is dead...",
  "type": "external"
},
{
  "name": "saved-town",
  "description": "The town is saved! But at what cost
    ...",
  "type": "external"
},
    {
      "name": "bound",
      "description": "%1$s is bound",
      "type": "external"
    },
    {
      "name": "seen-dead-people",
      "description": "%1$s has seen dead people",
      "type": "internal"
    },
    {
      "name": "free",
      "description": "%1$s is free",
      "type": "internal"
    },
    {
      "name": "emerged",
      "description": "%1$s has emerged",
      "type": "internal"
    }
  ],
  "objects": [
    {
      "name": "player",
      "description": "the protagonist",
      "type": "character"
    },
    {
      "name": "meta",
      "description": "the author",
      "type": "character"
    },
    {
      "name": "sheriff",
      "description": "Joseph Mills",
      "type": "character"
    },
    {
      "name": "mayor",
      "description": "Bruce Ableton",
      "type": "character"
    },
{
  "name": "groundskeeper",
  "description": "Vivienne West",
  "type": "character"
},
{
  "name": "citizen-one",
  "description": "Jack Myers",
  "type": "character"
},
{
  "name": "citizen-two",
  "description": "Jill Mills",
  "type": "character"
},
{
  "name": "farmer",
  "description": "Matthias Cooper",
  "type": "character"
},
{
  "name": "doctor",
  "description": "William Archibald Barrington",
  "type": "character"
},
{
  "name": "storeowner",
  "description": "Emilia Brooks",
  "type": "character"
},
{
  "name": "mindworm",
  "description": "a worm",
  "type": "character"
},
{
  "name": "general-store",
  "description": "the general store",
  "type": "location",
  "long-context": "Looking up, you can see a crooked
     sign hanging across the entrance that reads: '
     Everhill General Store'. The doorknob is slightly
```

```
      loose, and at close inspection you notice the
     paint on the door has started to peel. Upon
     opening the door, you can see the shelves are
     full of fresh produce from Cooper Farms. On the
     counter, you see a simple credit card machine and
      a tip jar, the bottom of which is lined with
     dollar bills.",
  "short-context": "Turning the loose doorknob, you
     are greeted once more by the familiar sights of
     the general store."
},
{
  "name": "town-hall",
  "description": "the town hall",
  "type": "location",
  "long-context": "The first thing you see upon
     arriving at the town hall is the well kept garden
      of Mayor Ableton. The double doors leading into
     the town hall are made of mahogony, and the brass
      doorhandles were recently polished. The front of
      the town hall has a fresh coat of paint, but
     around the edges of the walls, you can see paint
     peeling on the sides. When you walk in, you are
     immediately greeted with piles of paperwork
     scattered across the desk, almost haphazardly.",
  "short-context": "Opening the doors of the town hall
     , you are once again greeted by piles of
     paperwork."
},
{
  "name": "home",
  "description": "your home",
  "type": "location",
  "long-context": "There is something very comforting
     about being in your own home. Even though you
     live alone, you always feel like you're being
     welcomed back here. The house is in pretty good
     condition, although it wouldn't hurt to tidy up a
      bit. You don't have a lot here, but you have
     what you need, and the general store is just down
      the block if you find yourself wanting for
     something.",
  "short-context": "Even in tough times, you're always
      happy to welcome yourself back home."
},
{
  "name": "house-one",
  "description": "Worker's Respite",
  "type": "location",
  "long-context": "As you show up to the bed and
     breakfast, you see that it has recently gotten a
     fresh coat of beige paint. The immaculate sign
     hanging over the door shows the words 'The Worker
     's Respite' above a fine painting of a bed. On
     the top of the building, you can see a satellite
     dish, which appears to be a recent addition.",
  "short-context": "Upon returning to the Worker's
     Respite, you are once more greeted with the
     immaculate sign."
},
{
  "name": "house-two",
  "description": "Jill's house",
  "type": "location",
  "long-context": "As you approach the house of the
     Mills family, the first thing you see is a car
     parked out front. The hood is propped open, and a
      box of tools is resting next to the front left
     tire. The smell of gasoline and machine oil is
     abundant, and the ground is covered with grease
     stains. The house itself is a modest, two-story
     home, with three windows visible from the front
     of the building. The front is painted a faint
     green, and the door is unlocked.",
  "short-context": "Entering the vicinity of the Mills
      household once more, you can clearly smell the
     scent of machine oil."
},
{
  "name": "hospital",
  "description": "the hospital",
  "type": "location",
  "long-context": "You step through the clean glass
     doors of Everhill Medical Center into a small
```

```
      lobby. The waiting area is nearly untouched,
      except for a single magazine that somehow found
      its way onto the floor. As usual, there is nobody
       waiting to see the doctor, and the check-in
      sheet lists only a single appointment scheduled
      for later in the day. The air smells a bit like
      rubbing alcohol and medicine.",
    "short-context": "The scent of medicine welcomes you
       back to the still quiet lobby of Everhill
      Medical Center."
  },
  {
    "name": "farm",
    "description": "the farm",
    "type": "location",
    "long-context": "You approach a barn and a small,
      worn house you know to be the home of Farmer
      Cooper. Beyond a fence of wood and wire, you see
      the sprawling fields of grain and produce that
      regularly stock the town's general store. It
      looks like a great deal of care is put into
      maintaining these lands. The open fields manage
      to be both peaceful and almost uncomfortably
      quiet.",
    "short-context": "You walk back into the open fields
       of Cooper Farm. The only sounds you hear are the
       wind and the occasional creak of the barn door."
  },
  {
    "name": "sheriff-office",
    "description": "the sheriff's office",
    "type": "location",
    "long-context": "The first thing you smell as you
      approach the sheriff's office is the scent of
      freshly baked cookies. The worn, but well kept
      wooden door is propped open a few inches, and it
      is clear that the room inside is well lit. As you
       walk through the door, you notice that the front
       desk has several \"Mills for Mayor\" pins laying
       on top of it, almost haphazardly.",
    "short-context": "As you enter the sheriff's office,
       the old door welcomes you back with a creak as
      it opens."
  },
  {
    "name": "knife",
    "description": "a knife",
    "type": "item"
  },
  {
    "name": "gun",
    "description": "a gun",
    "type": "item"
  },
  {
    "name": "pitchfork",
    "description": "a pitchfork",
    "type": "item"
  },
  {
    "name": "gasoline",
    "description": "gasoline",
    "type": "item"
  },
  {
    "name": "alcohol",
    "description": "alcohol",
    "type": "item"
  },
  {
    "name": "handcuffs",
    "description": "a pair of handcuffs",
    "type": "item"
  },
  {
    "name": "duct-tape",
    "description": "a roll of duct tape",
    "type": "item"
  },
  {
    "name": "lighter",
    "description": "a lighter",
    "type": "item"
  }
```

```
    ]
}
```

D Graphics Decoration File for *The Worm of Everhill*

```
{
  "actions" : [
    {
      "name" : "wait",
      "description" : "%1$s waits",
      "command" : "wait",
      "type" : "other",
      "panelType" : "2panel",
      "imgPath" : "Stories/Horror/Assets/temp.png",
      "importantTerms" : "0",
      "actionNeeded" : "false"
    },
    {
      "name" : "move",
      "command" : "move to %3$s",
      "type" : "move",
      "panelType" : "1panel",
      "imgPath" : "Stories/Horror/Assets/move.png",
      "importantTerms" : "1",
      "actionNeeded" : "false",
      "description": [
        "%1$s moves to %3$s",
        "%1$s travels to %3$s",
        "%1$s jogs to %3$s",
        "%1$s walks to %3$s",
        "%1$s runs to %3$s"
      ],
      "variant-argument" : 1,
      "variants" : [
        {
          "key" : "mindworm",
          "description" : "%1$s quickly crawls off towards
              %3$s",
          "command" : "move to %3$s"
        }
      ]
    },
    {
```

```
    "name" : "target-human",
    "description" : "%1$s approaches %2$s menacingly",
    "command" : "approach %2$s menacingly",
    "type" : "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/temp.png",
    "importantTerms" : "0",
    "actionNeeded" : "false",
    "variant-argument" : 3,
    "variants" : [
      {
        "key" : "gun",
        "description" : "%1$s lines up a shot towards %2
           $s",
        "command" : "aim at %2$s"
      },
      {
        "key" : "pitchfork",
        "description" : "%1$s approaches %2$s menacingly
           , pitchfork held high",
        "command" : "approach %2$s, pitchfork raised"
      }
    ]
  },
  {
    "name" : "kill",
    "description" : "%1$s kills %2$s with %4$s",
    "command" : "kill %2$s with %4$s",
    "type" : "other",
    "panelType" : "3panel_leftquarters",
    "imgPath" : "Stories/Horror/Assets/kill.png",
    "importantTerms" : "1 4 2",
    "actionNeeded" : "false"
  },
  {
    "name": "kill-possessed-human",
    "description": "%1$s kills %2$s with %5$s",
    "command": "kill %2$s with %5$s",
    "type": "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/kill.png",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  },
  {
    "name" : "kill-self",
    "description" : "%1$s commits suicide with %2$s",
    "command" : "commit suicide with %2$s",
    "type" : "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/kill.png",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  },
  {
    "name" : "loot-body",
    "description" : "%1$s loots %3$s from %2$s's body",
    "command" : "loot %3$s from %2$s's body",
    "type" : "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/loot.png",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  },
  {
    "name": "player-loot-body",
    "description": "%1$s loots %3$s from %2$s's body",
    "command": "loot %3$s from %2$s's body",
    "type": "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/loot.png",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  },
  {
    "name" : "pickup",
    "description" : "%1$s picks up %2$s",
    "command" : "pick up %2$s",
    "type" : "other",
    "panelType" : "3panel",
    "imgPath" : "Stories/Horror/Assets/pickup.jpg",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  },
  {
    "name": "player-pickup",
    "description": "%1$s picks up %2$s",
    "command": "pick up %2$s",
    "type": "other",
    "panelType" : "3panel",
    "imgPath" : "Stories/Horror/Assets/pickup.jpg",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  },
  {
    "name" : "putdown",
    "description" : "%1$s puts down %2$s",
    "command" : "put down %2$s",
    "type" : "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/putdown.png",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  },
  {
    "name" : "douse-person",
    "description" : "%1$s douses %2$s in %3$s",
    "command" : "douse %2$s with %3$s",
    "type" : "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/douse-person.png
       ",
    "importantTerms" : "3 2",
    "actionNeeded" : "false"
  },
  {
    "name" : "ignite",
    "description" : "%1$s lights %2$s on fire with %4$s
       ",
    "command" : "light %2$s on fire with %4$s",
    "type" : "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/ignite.png",
    "importantTerms" : "4 2",
    "actionNeeded" : "false"
  },
  {
    "name" : "ignite-possessed",
    "description" : "%1$s lights %2$s on fire with %5$s
       ",
    "command" : "light %2$s on fire with %5$s",
    "type" : "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/ignite.png",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  },
  {
    "name" : "possess-human",
    "description" : "%1$s enters %2$s through the ear
       canal",
    "command" : "enter %2$s through the ear canal",
    "type" : "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/possess-human.png
       ",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  },
  {
    "name" : "leave-possessed-human",
    "description" : "%1$s erupts from %2$s nostril",
    "command" : "erupt from %2$s nostril",
    "type" : "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/leave-possessed-
       human.png",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  },
  {
    "name" : "talk-to-human",
    "description" : "%1$s has a pleasant conversation
       with %2$s",
    "command" : "Talk to %2$s",
    "type" : "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/temp.png",
    "importantTerms" : "1 2",
    "actionNeeded" : "false"
  },
  {
    "name" : "talk-to-possessed-human",
    "description" : "%1$s tries to start a conversation
       with %2$s, but %2$s remains eerily silent",
    "command" : "Talk to %2$s",
    "type" : "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/temp.png",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  },
  {
    "name": "bind",
    "description": "%1$s restrains %2$s using the %3$s",
    "command": "Restrain %2$s with %3$s",
    "type": "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/temp.png",
    "importantTerms" : "3 2",
    "actionNeeded" : "false"
  },
  {
    "name": "bind-possessed-human",
    "description": "%1$s restrains %2$s using the %4$s,
       and a worm erupts from %2$s's nostril",
    "command": "Restrain %2$s with %4$s",
    "type": "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/temp.png",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  },
  {
    "name": "emerge",
    "description": "%1$s emerges into %2$s",
    "command": "Emerge",
    "type": "other",
    "panelType" : "2panel",
    "imgPath" : "Stories/Horror/Assets/temp.png",
    "importantTerms" : "0",
    "actionNeeded" : "false"
  }
],

"predicates" : [
  {
    "name" : "dead",
    "description" : "%1$s is dead",
    "type" : "external",
    "renderType" : "characterState",
    "variant-argument" : 1,
    "variants" : [
      {
        "key" : "player",
        "description" : "You are left laying in a pool
           of your own blood. Flies begin to surround
           your fresh wounds as a metallic odor tinges
           the air, and your body lies cold and
           motionless"
      },
      {
        "key" : "groundskeeper",
        "description" : "%1$s lies lifeless on the
           ground. She looks sad but strangely peaceful,
            her colorful dress drenched in blood"
      },
      {
        "key" : "mayor",
        "description" : "%1$s's body lays in a heap. A
           campaign pin, \"Ableton for mayor\", lies at
           his side, stained with blood"
      },
      {
        "key" : "farmer",
        "description" : "%1$s is dead, but his eyes
           remain open. His hand is twisted, almost
           strained, as if grasping for something"
      },
      {
        "key" : "citizen-one",
        "description" : "%1$s is dead, one hand
```

```
            clutching his chest , the other reaching out
            for something or someone . He was not ready to
             go"
        },
        {
          "key" : "citizen-two",
          "description" : "%1$s's body is soaked in blood ,
             laying still and breathless on the ground .
            She had put up a fight"
        },
        {
          "key" : "doctor",
          "description" : "%1$s lays dead on the ground ,
            blood pooling around him . His teeth are bared
             in a grimace"
        },
        {
          "key" : "storeowner",
          "description" : "%1$s's body lays quiet . She is
            dead . She was the oldest person in town , but
            she couldn't survive this day"
        }
      ]
    },
    {
      "name" : "at",
      "description": [
        "%1$s is at %2$s"
      ],
      "type" : "locational",
      "renderType" : "locationState"
    },
    {
      "name" : "armed",
      "description" : "%1$s is armed",
      "type" : "external",
      "renderType" : "noRender"
    },
    {
      "name" : "has",
      "description" : "%1$s has %2$s",
      "type" : "external",
  "renderType" : "itemState"
},
{
  "name" : "controlling",
  "description" : "%1$s is controlling %2$s",
  "type" : "internal",
  "renderType" : "noRender"
},
{
  "name" : "possessed",
  "description" : "%1$s is possessed",
  "type" : "internal",
  "renderType" : "noRender"
},
{
  "name" : "possessing",
  "description" : "%1$s is possessing a human",
  "type" : "internal",
  "renderType" : "noRender"
},
{
  "name" : "suspicious",
  "description" : "%1$s is acting suspiciously",
  "type" : "external",
  "renderType" : "characterState"
},
{
  "name" : "doused",
  "description" : "%1$s is doused in a flammable
     liquid",
  "type" : "external",
  "renderType" : "characterState"
},
{
  "name" : "scorched",
  "description" : "%1$s's body is burned to a crisp",
  "type" : "external",
  "renderType" : "characterState"
},
{
  "name" : "targeting",
  "description" : "%1$s is focused on %2$s",
    "type" : "internal",
    "renderType" : "characterState"
  },
  {
    "name" : "goal-achieved",
    "description" : "The story is complete",
    "type" : "external",
    "renderType" : "noRender"
  },
  {
    "name" : "destroyed-town",
    "description" : "Everyone is dead...",
    "type" : "external",
    "renderType" : "noRender"
  },
  {
    "name" : "saved-town",
    "description" : "The town is saved! But at what cost
       ...",
    "type" : "external",
    "renderType" : "noRender"
  },
  {
    "name": "bound",
    "description": "%1$s is bound",
    "type": "external",
    "renderType" : "noRender"
  },
  {
    "name": "seen-dead-people",
    "description": "%1$s has seen dead people",
    "type": "hidden",
    "renderType" : "noRender"
  },
  {
    "name": "free",
    "description": "%1$s is free",
    "type": "internal",
    "renderType" : "noRender"
  },
  {
    "name": "emerged",
    "description": "%1$s has emerged",
    "type": "internal",
    "renderType" : "noRender"
  }
],

"objects": [
  {
    "name": "player",
    "description": "the protagonist",
    "type": "character"
  },
  {
    "name": "meta",
    "description": "the author",
    "type": "character"
  },
  {
    "name": "sheriff",
    "description": "Joseph Mills",
    "type": "character"
  },
  {
    "name": "mayor",
    "description": "Bruce Ableton",
    "type": "character"
  },
  {
    "name": "groundskeeper",
    "description": "Vivienne West",
    "type": "character"
  },
  {
    "name": "citizen-one",
    "description": "Jack Myers",
    "type": "character"
  },
  {
    "name": "citizen-two",
    "description": "Jill Mills",
    "type": "character"
  },
{
  "name": "farmer",
  "description": "Matthias Cooper",
  "type": "character"
},
{
  "name": "doctor",
  "description": "William Archibald Barrington",
  "type": "character"
},
{
  "name": "storeowner",
  "description": "Emilia Brooks",
  "type": "character"
},
{
  "name": "mindworm",
  "description": "a worm",
  "type": "character"
},
{
  "name": "general-store",
  "description": "the general store",
  "type": "location",
  "long-context": "Looking up, you can see a crooked
     sign hanging across the entrance that reads: '
     Everhill General Store'. The doorknob is slightly
      loose, and at close inspection you notice the
     paint on the door has started to peel. Upon
     opening the door, you can see the shelves are
     full of fresh produce from Cooper Farms. On the
     counter, you see a simple credit card machine and
      a tip jar, the bottom of which is lined with
     dollar bills.",
  "short-context": "Turning the loose doorknob, you
     are greeted once more by the familiar sights of
     the general store."
},
{
  "name": "town-hall",
  "description": "the town hall",
  "type": "location",
  "long-context": "The first thing you see upon
     arriving at the town hall is the well kept garden
      of Mayor Ableton. The double doors leading into
     the town hall are made of mahogony, and the brass
      doorhandles were recently polished. The front of
      the town hall has a fresh coat of paint, but
     around the edges of the walls, you can see paint
     peeling on the sides. When you walk in, you are
     immediately greeted with piles of paperwork
     scattered across the desk, almost haphazardly.",
  "short-context": "Opening the doors of the town hall
     , you are once again greeted by piles of
     paperwork."
},
{
  "name": "home",
  "description": "your home",
  "type": "location",
  "long-context": "There is something very comforting
     about being in your own home. Even though you
     live alone, you always feel like you're being
     welcomed back here. The house is in pretty good
     condition, although it wouldn't hurt to tidy up a
      bit. You don't have a lot here, but you have
     what you need, and the general store is just down
      the block if you find yourself wanting for
     something.",
  "short-context": "Even in tough times, you're always
      happy to welcome yourself back home."
},
{
  "name": "house-one",
  "description": "Worker's Respite",
  "type": "location",
  "long-context": "As you show up to the bed and
     breakfast, you see that it has recently gotten a
     fresh coat of beige paint. The immaculate sign
     hanging over the door shows the words 'The Worker
     's Respite' above a fine painting of a bed. On
     the top of the building, you can see a satellite
     dish, which appears to be a recent addition.",
  "short-context": "Upon returning to the Worker's
```

```
      Respite, you are once more greeted with the
      immaculate sign."
  },
  {
    "name": "house-two",
    "description": "Jill's house",
    "type": "location",
    "long-context": "As you approach the house of the
      Mills family, the first thing you see is a car
      parked out front. The hood is propped open, and a
       box of tools is resting next to the front left
      tire. The smell of gasoline and machine oil is
      abundant, and the ground is covered with grease
      stains. The house itself is a modest, two-story
      home, with three windows visible from the front
      of the building. The front is painted a faint
      green, and the door is unlocked.",
    "short-context": "Entering the vicinity of the Mills
       household once more, you can clearly smell the
      scent of machine oil."
  },
  {
    "name": "hospital",
    "description": "the hospital",
    "type": "location",
    "long-context": "You step through the clean glass
      doors of Everhill Medical Center into a small
      lobby. The waiting area is nearly untouched,
      except for a single magazine that somehow found
      its way onto the floor. As usual, there is nobody
       waiting to see the doctor, and the check-in
      sheet lists only a single appointment scheduled
      for later in the day. The air smells a bit like
      rubbing alcohol and medicine.",
    "short-context": "The scent of medicine welcomes you
       back to the still quiet lobby of Everhill
      Medical Center."
  },
  {
    "name": "farm",
    "description": "the farm",
    "type": "location",
  "long-context": "You approach a barn and a small,
     worn house you know to be the home of Farmer
     Cooper. Beyond a fence of wood and wire, you see
     the sprawling fields of grain and produce that
     regularly stock the town's general store. It
     looks like a great deal of care is put into
     maintaining these lands. The open fields manage
     to be both peaceful and almost uncomfortably
     quiet.",
  "short-context": "You walk back into the open fields
      of Cooper Farm. The only sounds you hear are the
      wind and the occasional creak of the barn door."
},
{
  "name": "sheriff-office",
  "description": "the sheriff's office",
  "type": "location",
  "long-context": "The first thing you smell as you
     approach the sheriff's office is the scent of
     freshly baked cookies. The worn, but well kept
     wooden door is propped open a few inches, and it
     is clear that the room inside is well lit. As you
      walk through the door, you notice that the front
      desk has several \"Mills for Mayor\" pins laying
      on top of it, almost haphazardly.",
  "short-context": "As you enter the sheriff's office,
      the old door welcomes you back with a creak as
     it opens."
},
{
  "name": "knife",
  "description": "a knife",
  "type": "item"
},
{
  "name": "gun",
  "description": "a gun",
  "type": "item"
},
{
  "name": "pitchfork",
  "description": "a pitchfork",
    "type": "item"
  },
  {
    "name": "gasoline",
    "description": "gasoline",
    "type": "item"
  },
  {
    "name": "alcohol",
    "description": "alcohol",
    "type": "item"
  },
  {
    "name": "handcuffs",
    "description": "a pair of handcuffs",
    "type": "item"
  },
  {
    "name": "duct-tape",
    "description": "a roll of duct tape",
    "type": "item"
  },
  {
    "name": "lighter",
    "description": "a lighter",
    "type": "item"
  }
],

"locations" : [
  {
    "name" : "general-store",
    "description" : "the general store",
    "type"  : "location",
    "long-context" : "Looking up, you can see a crooked
       sign hanging across the entrance that reads: '
       Everhill General Store'. The doorknob is slightly
        loose, and at careful look the paint on the door
        has started to peel. On opening the door, you
       can see the shelves are full of fresh produce
       from Cooper Farms. On the counter, you see a
       simple credit card machine and a tip jar, the
```

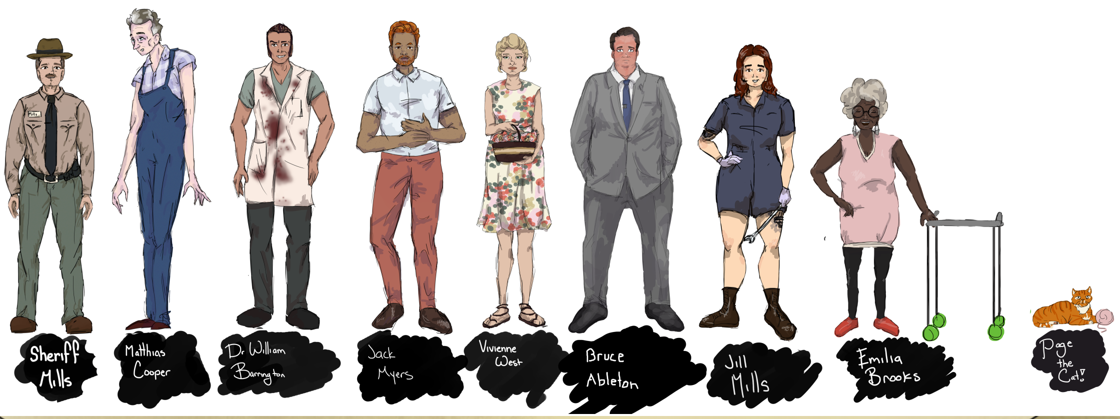
```
      bottom of which is lined with dollar bills.",
    "short-context" : "Turning the loose doorknob, you
      are greeted once more by the familiar sights of
      the general store.",
    "image-path" : "Stories/Horror/Assets/Locations/
      general_store.png",
    "small-item-slots" : "130 205",
    "large-item-slots" : "110 75",
    "character-slots" : "30 30, 420 40, 250 20",
    "actionImagePath" : "Stories/Horror/Assets/Locations
      /general_store.png"
  },
  {
    "name" : "town-hall",
    "description" : "the town hall",
    "type"  : "location",
    "long-context" : "The first thing you see on
      arriving at the town hall is the well kept garden
       of Mayor Ableton. The double doors leading into
      the town hall are made of mahogony, and the brass
       doorhandles were recently polished. The front of
       the town hall has a fresh coat of paint, but
      around the edges of the walls, you can see paint
      peeling on the sides. On walking in, you are
      immediately greeted with piles of paperwork
      scattered across the desk, almost haphazardly.",
    "short-context" : "Opening the doors once more, you
      are once again greeted by piles of paperwork.",
    "image-path" : "Stories/Horror/Assets/Locations/
      town_hall.png",
    "small-item-slots" : "300 265, 350 265",
    "large-item-slots" : "700 100",
    "character-slots" : "30 30, 420 40",
    "actionImagePath" : "Stories/Horror/Assets/Locations
      /town_hall.png"
  },
  {
    "name" : "home",
    "description" : "the protagonist's home",
    "type"  : "location",
    "long-context" : "There is something very comforting
       about being in your own home. Even though you
```

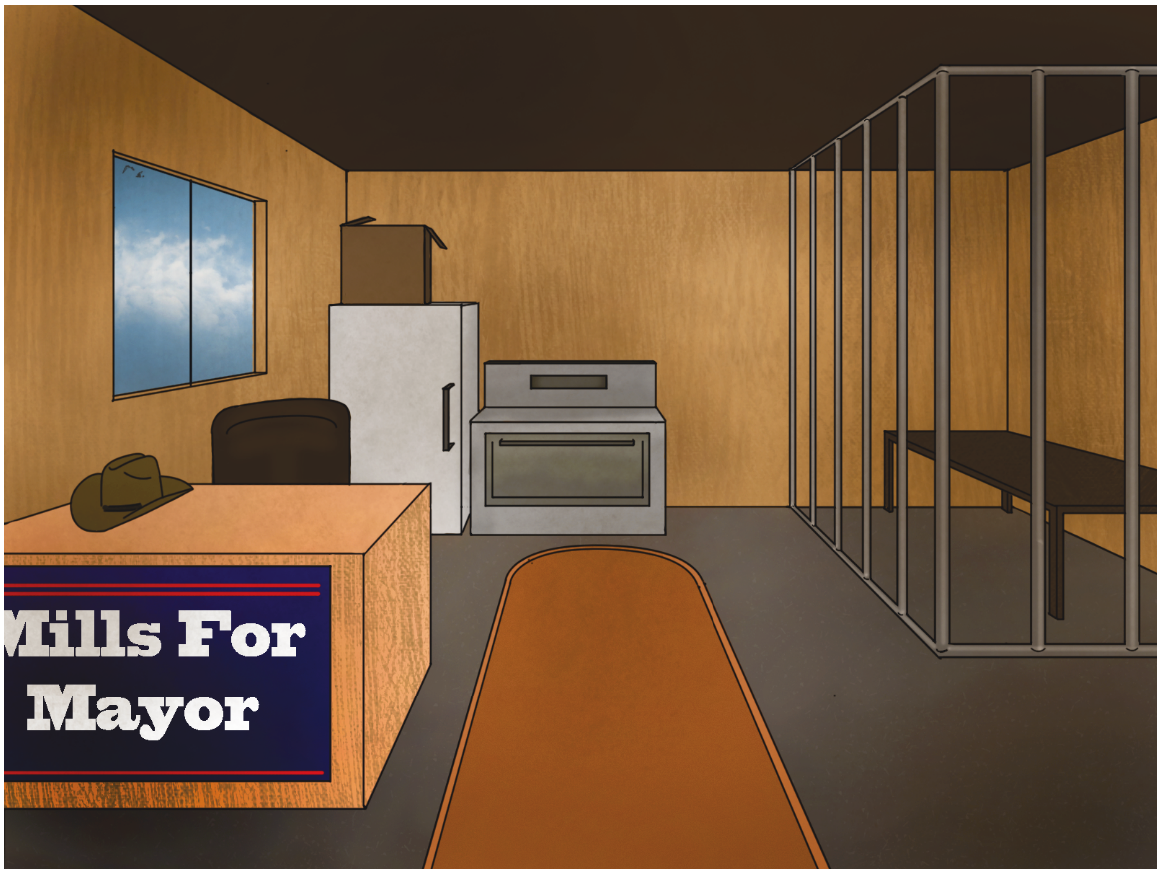
```
        live alone, you always feel like you're being
        welcomed back here. The house is in pretty good
        condition, although you could always use to tidy
        up a bit. You don't have a lot here, but you have
         what you need, and the general store is just
        down the block if you find yourself wanting for
        something.",
    "short-context" : "Even in tough times, you're
        always happy to welcome yourself back home.",
    "image-path" : "Stories/Horror/Assets/Locations/home
        .png",
    "small-item-slots" : "420 170, 390 100",
    "large-item-slots" : "260 215",
    "character-slots" : "30 10, 650 30, 550 150",
    "actionImagePath" : "Stories/Horror/Assets/Locations
        /home.png"
},
{
    "name" : "house-one",
    "description" : "Worker's Respite",
    "type"  : "location",
    "long-context" : "On showing up to bed and breakfast
        , you see that it has recently gotten a fresh
        coat of beige paint. The immaculate sign hanging
        over the door shows the words 'The Worker's
        Respite' above a fine painting of of a bed. On
        the top of a building, you can see a satallite
        dish, which appears to be a recent addition to
        the building.",
    "short-context" : "Upon returning to the Worker's
        Respite, you are once more greeted with the
        immaculate sign.",
    "image-path" : "Stories/Horror/Assets/Locations/
        workers_respite.png",
    "small-item-slots" : "350 120, 300 50, 415 50",
    "large-item-slots" : "400 300",
    "character-slots" : "300 30, 550 30, 500 30, 600
        30",
    "actionImagePath" : "Stories/Horror/Assets/Locations
        /workers_respite.png"
},
{
    "name" : "house-two",
    "description" : "Jill's house",
    "type"  : "location",
    "long-context" : "As you approach the house of the
        Mills family, the first thing you see is a car
        parked out front. The hood is propped open, and a
         box of tools is resting next to the front left
        tire. The smell of gasoline and machine oil is
        clear to smell, and the ground is covered with
        grease stains. The house itself is a modest, two-
        story home, with three windows visible from the
        front of the building. The front is painted a
        faint green, and the door is unlocked.",
    "short-context" : "Entering the vicinity of the
        Mills household once more, you can clearly smell
        the scent of machine oil.",
    "image-path" : "Stories/Horror/Assets/Locations/
        mills_house.png",
    "small-item-slots" : "200 200, 640 280",
    "large-item-slots" : "450 225",
    "character-slots" : "75 75, 450 30",
    "actionImagePath" : "Stories/Horror/Assets/Locations
        /mills_house.png"
},
{
    "name" : "hospital",
    "description" : "the hospital",
    "type"  : "location",
    "long-context" : "You step through the clean glass
        doors of Everhill Medical Center into a small
        lobby. The waiting area is nearly untouched,
        except for a single magazine that somehow found
        its way onto the floor. As usual, there is nobody
         waiting to see the doctor, and the check-in
        sheet lists only a single appointment scheduled
        for later that day. The air smells a bit like
        rubbing alcohol and medicine.",
    "short-context" : "The scent of medicine welcomes
        you back to the still quiet lobby of Everhill
        Medical Center.",
    "image-path" : "Stories/Horror/Assets/Locations/
        hospital.png",
    "small-item-slots" : "300 300, 200 200",
    "large-item-slots" : "100 75",
    "character-slots" : "300 300, 200 200",
    "actionImagePath" : "Stories/Horror/Assets/Locations
         /hospital.png"
},
{
    "name" : "farm",
    "description" : "the farm",
    "type"  : "location",
    "long-context" : "You approach a barn and a small,
       worn house you know to be the home of Farmer
       Cooper. Beyond a fence of wood and wire, you see
       the sprawling fields of grain and produce that
       regularly stock the town's general store. It
       looks like a great deal of care is put into
       maintaining these lands. The open fields manage
       to be both peaceful and almost uncomfortably
       quiet.",
    "short-context" : "You walk back into the open
       fields of Cooper Farm. The only sounds you hear
       are the wind and the occasional creak of the barn
        door.",
    "image-path" : "Stories/Horror/Assets/Locations/farm
       .png",
    "small-item-slots" : "570 140, 10 150",
    "large-item-slots" : "100 110",
    "character-slots" : "175 20, 400 35",
    "actionImagePath" : "Stories/Horror/Assets/Locations
       /farm.png"
},
{
    "name" : "sheriff-office",
    "description" : "the sheriff's office",
    "type"  : "location",
    "long-context" : "The first thing you smell as you
       approach the sheriff's office is the smell of
       freshly baked cookies. The worn, but well kept
       wooden door is propped open a few inches, and it
       is clear the room inside is well lit. As you walk
        inside the door, you notice that the front desk
       has several Mills for Mayor pins lying on top of
```

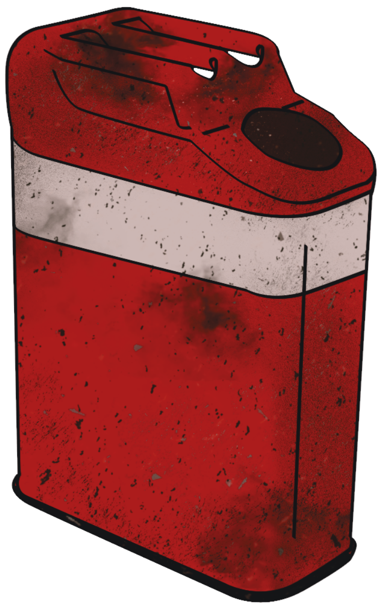
```
            it, almost haphazardly. The small cell in the
            corner is empty, and a spare pair of handcuff's
            looks as though it has never been used.",
        "short-context" : "As you enter the sheriff's office
            , you once again hear the welcoming creak of the
            door as it opens.",
        "image-path" : "Stories/Horror/Assets/Locations/
            sheriff-office.png",
        "small-item-slots" : "140 120, 160 120",
        "large-item-slots" : "100 75",
        "character-slots" : "30 30, 300 70",
        "actionImagePath" : "Stories/Horror/Assets/Locations
            /sheriff-office.png"
    }
],

"items" : [
    {
        "name" : "knife",
        "description" : "a knife",
        "type"  : "small",
        "size" : "50 50",
        "image-path" : "Stories/Horror/Assets/Objects/knife.
            png",
        "P1" : "0 0",
        "P2" : "50 50"
    },
    {
        "name" : "gun",
        "description" : "a gun",
        "type"  : "small",
        "size" : "72 51",
        "image-path" : "Stories/Horror/Assets/Objects/gun.
            png",
        "P1" : "0 0",
        "P2" : "50 50"
    },
    {
        "name" : "pitchfork",
        "description" : "a pitchfork",
        "type"  : "large",
        "size" : "220 220",
    "image-path" : "Stories/Horror/Assets/Objects/
       pitchfork.png",
    "P1" : "0 0",
    "P2" : "50 50"
  },
  {
    "name" : "gasoline",
    "description" : "gasoline",
    "type"  : "large",
    "size" : "-72 90",
    "image-path" : "Stories/Horror/Assets/Objects/
       gasoline.png",
    "P1" : "0 0",
    "P2" : "50 50"
  },
  {
    "name" : "alcohol",
    "description" : "alcohol",
    "type"  : "small",
    "size" : "40 90",
    "image-path" : "Stories/Horror/Assets/Objects/
       alcohol.png",
    "P1" : "0 0",
    "P2" : "50 50"
  },
  {
    "name" : "handcuffs",
    "description" : "a pair of handcuffs",
    "type"  : "small",
    "size" : "70 70",
    "image-path" : "Stories/Horror/Assets/Objects/
       handcuffs.png",
    "P1" : "0 0",
    "P2" : "50 50"
  },
  {
    "name" : "duct-tape",
    "description" : "a roll of duct tape",
    "type"  : "small",
    "size" : "50 40",
    "image-path" : "Stories/Horror/Assets/Objects/
       duct_tape.png",
      "P1" : "0 0",
      "P2" : "50 50"
    },
    {
      "name" : "lighter",
      "description" : "a lighter",
      "type"  : "small",
      "size" : "10 15",
      "image-path" : "Stories/Horror/Assets/Objects/
         lighter.png",
      "P1" : "0 0",
      "P2" : "50 50"
    }
  ],

  "characters" : [
    {
      "name" : "player",
      "description" : "the protagonist",
      "type"  : "character",
      "size" : "150 400",
      "image-path" : "Stories/Horror/Assets/player.png",
      "R1" : "40 138",
      "R2" : "120 188",
      "L1" : "90 150",
      "L2" : "100 160",
      "states" : [
        {
          "state1Name" : "normal",
          "state1Path" : "Stories/Horror/Assets/player.png
             "
        },
        {
          "state2Name" : "dead",
          "state2Path" : "Stories/Horror/Assets/player.png
             ",
        }]
    },
    {
      "name" : "meta",
      "description" : "the author",
      "type"  : "character",
    "size" : "0 0",
    "image-path" : "Stories/Horror/Assets/temp.png",
    "R1" : "10 150",
    "R2" : "0 160",
    "L1" : "90 150",
    "L2" : "100 160",
    "states" : [
      {
        "state1Name" : "normal",
        "state1Path" : "Stories/Horror/Assets/temp.png"
      }]
  },
  {
    "name" : "sheriff",
    "description" : "the sheriff",
    "type"  : "character",
    "size" : "150 400",
    "image-path" : "Stories/Horror/Assets/Characters/
       SheriffMills.png",
    "R1" : "10 150",
    "R2" : "0 160",
    "L1" : "90 150",
    "L2" : "100 160",
    "states" : [
      {
        "state1Name" : "normal",
        "state1Path" : "Stories/Horror/Assets/Characters
           /SheriffMills.png",
      },
      {
        "state2Name" : "dead",
        "state2Path" : "Stories/Horror/Assets/Characters
           /SheriffMills.png",
      }],
  },
  {
    "name" : "mayor",
    "description" : "the mayor",
    "type"  : "character",
    "size" : "150 400",
    "image-path" : "Stories/Horror/Assets/Characters/
       BruceAbleton.png",
    "R1" : "10 150",
    "R2" : "0 160",
    "L1" : "90 150",
    "L2" : "100 160",
    "states" : [
      {
        "state1Name" : "normal",
        "state1Path" : "Stories/Horror/Assets/Characters
           /BruceAbleton.png",
      },
      {
        "state2Name" : "dead",
        "state2Path" : "Stories/Horror/Assets/Characters
           /BruceAbleton.png",
      }],
  },
  {
    "name" : "groundskeeper",
    "description" : "Vivienne West",
    "type"  : "character",
    "size" : "150 400",
    "image-path" : "Stories/Horror/Assets/Characters/
       Vivienne.png",
    "R1" : "10 150",
    "R2" : "0 160",
    "L1" : "90 150",
    "L2" : "100 160",
    "states" : [
      {
        "state1Name" : "normal",
        "state1Path" : "Stories/Horror/Assets/Characters
           /Vivienne.png",
      },
      {
        "state2Name" : "dead",
        "state2Path" : "Stories/Horror/Assets/Characters
           /Vivienne.png",
      }],
  },
  {
    "name" : "citizen-one",
    "description" : "Jack",
    "type"  : "character",
    "size" : "150 400",
    "image-path" : "Stories/Horror/Assets/Characters/
       JackMyers.png",
    "R1" : "10 150",
    "R2" : "0 160",
    "L1" : "90 150",
    "L2" : "100 160",
    "states" : [
      {
        "state1Name" : "normal",
        "state1Path" : "Stories/Horror/Assets/Characters
           /JackMyers.png",
      },
      {
        "state2Name" : "dead",
        "state2Path" : "Stories/Horror/Assets/Characters
           /JackMyers.png",
      }],
  },
  {
    "name" : "citizen-two",
    "description" : "Jill",
    "type"  : "character",
    "size" : "180 400",
    "image-path" : "Stories/Horror/Assets/Characters/
       Jill.png",
    "R1" : "10 150",
    "R2" : "0 160",
    "L1" : "90 150",
    "L2" : "100 160",
    "states" : [
      {
        "state1Name" : "normal",
        "state1Path" : "Stories/Horror/Assets/Characters
           /Jill.png",
      },
      {
        "state2Name" : "dead",
        "state2Path" : "Stories/Horror/Assets/Characters
           /Jill.png",
      }],
  },
  {
    "name" : "farmer",
    "description" : "the farmer",
    "type"  : "character",
    "size" : "150 400",
    "image-path" : "Stories/Horror/Assets/Characters/
       Matthias.png",
    "R1" : "10 150",
    "R2" : "0 160",
    "L1" : "90 150",
    "L2" : "100 160",
    "states" : [
      {
        "state1Name" : "normal",
        "state1Path" : "Stories/Horror/Assets/Characters
           /Matthias.png",
      },
      {
        "state2Name" : "dead",
        "state2Path" : "Stories/Horror/Assets/Characters
           /Matthias.png",
      }],
  },
  {
    "name" : "doctor",
    "description" : "the doctor",
    "type"  : "character",
    "size" : "150 400",
    "image-path" : "Stories/Horror/Assets/Characters/
       Doctor.png",
    "R1" : "10 150",
    "R2" : "0 160",
    "L1" : "90 150",
    "L2" : "100 160",
    "states" : [
      {
        "state1Name" : "normal",
        "state1Path" : "Stories/Horror/Assets/Characters
           /Doctor.png",
      },
      {
        "state2Name" : "dead",
        "state2Path" : "Stories/Horror/Assets/Characters
           /Doctor.png",
      }],
  },
  {
    "name" : "storeowner",
    "description" : "Emilia",
    "type" : "character",
    "size" : "150 400",
    "image-path" : "Stories/Horror/Assets/Characters/
       Emilia.png",
    "R1" : "10 150",
    "R2" : "0 160",
    "L1" : "90 150",
    "L2" : "100 160",
    "states" : [
      {
        "state1Name" : "normal",
        "state1Path" : "Stories/Horror/Assets/Characters
           /Emilia.png",
      },
      {
        "state2Name" : "dead",
        "state2Path" : "Stories/Horror/Assets/Characters
           /Doctor.png",
      }],
  },
  {
    "name" : "mindworm",
    "description" : "a worm",
    "type"  : "character",
    "size" : "100 100",
    "image-path" : "Stories/Horror/Assets/Characters/
       mindworm.png",
    "R1" : "10 150",
    "R2" : "0 160",
    "L1" : "90 150",
    "L2" : "100 160",
    "states" : [
      {
        "state1Name" : "normal",
          "state1Path" : "Stories/Horror/Assets/Characters
            /mindworm.png",
        },
        {
          "state2Name" : "dead",
          "state2Path" : "Stories/Horror/Assets/Characters
            /mindworm.png",
        }],
    }
  ]
}
```

E Sample Art for *The Worm of Everhill*

All art for *The Worm of Everhill* was produced by Cherish Springer, advised by Professor Ed Gutierrez.




Mills For
Mayor



F Informed Consent Form for Playtesting

Introduction: You are being asked to participate in a research study on the development of digital games with procedurally generated narratives. Before you agree, however, you must be fully informed about the purpose of the study, the procedures to be followed, and any benefits, risks or discomfort that you may experience as a result of your participation. This form presents information about the study so that you may make a fully informed decision regarding your participation. Please feel free to pause and ask any questions you might have during or after reading this form.

Purpose of the study: The purpose of this study is to obtain playtest feedback on a digital game which dynamically generates narratives based on player actions, and to identify potential improvements in the user interface and narrative design.

Procedures to be followed: You will be asked to play a ten-minute session of a game. An investigator will record non-identifying data about the play session. After completing the game, you will be asked to complete a brief, anonymous survey describing your subjective experience.

Risks to study participants: The game being played contains non-graphic depictions of violence, murder, and immolation.

Benefits to research participants and others: You will have an opportunity to enjoy and comment on a new game under active development. Your feedback will help provide insight about the design of games using procedurally generated narrative.

Record keeping and confidentiality: Records of your participation in this study will be held confidential so far as permitted by law. However, the study investigators and, under certain circumstances, the Worcester Polytechnic Institute Institutional Review Board (WPI IRB) will be able to inspect and have access to confidential data that identify you by name. Any publication or presentation of the data will not identify you.

Compensation or treatment in the event of injury: There is no foreseeable risk of injury associated with this research study. Nevertheless, you do not give up any of your legal rights by signing this statement.

Your participation in this research is voluntary. Your refusal to participate will not result in any penalty to you or any loss of benefits to which you may otherwise be entitled. You may decide to stop participating in the research at any time without penalty or loss of other benefits. The project investigators retain the right to cancel or postpone the experimental procedures at any time they see fit.

By signing below, you acknowledge that you have been informed about and consent to be a participant in the study described above. Make sure that your questions are answered to your satisfaction before signing. You are entitled to retain a copy of this consent agreement.

G Scripted Introduction for Playtesting

Hello, and thank you for volunteering to test our game. Before we begin, could you please read and sign this Informed Consent form? *[Participant signs IC form.]* Thank you. During your test session, we will be recording data about your interaction with the game. You are encouraged to voice any thoughts you have during play. When your session is complete, we will ask you to complete a brief survey about your play experience. At no point during your play session, or in the survey after, will any sort of personal and/or identifying information about you be recorded. You may choose to stop playing and withdraw from the study at any point during the play session. Please begin playing when you feel ready.

H Survey Questions for Playtesting

- What is the objective of the game? How did you discover it?
- How difficult does the game seem?
- Did the game consistently hold your interest?
- Did anything about the game seem confusing or obscure?
- What would it have been good to know about the game before you started playing?
- How would you describe the game to someone who has never played it?
- Were you ever surprised about the actions available to you (e.g: having an action available to you which you did not expect to have, or an action unavailable to you which you expected to have)? Please explain.
- Were you ever surprised about the actions of other characters (e.g: a character acting in a way inconsistent with your expectations about the world and/or their personality)? Please explain.
- Was the method of input understandable and easy to use?
- Was the display of the game world or individual actions understandable and easy to use?
- Do you have any additional comments or questions that you would like to share?