University of Twente
Enschede - The Netherlands

University of Twente
Department of Electrical Engineering,
Mathematics & Computer Sciense
Faculty of Human Media Interaction

Wacky Pirate Hijinks
-
Causing and Resolving Conflict between Autonomous Agents in a Virtual Storyworld

February 26, 2009

Author:
Pjotter Tommassen

Examination Committee:
ir. I.M.T. Swartjes
dr. M. Theune
dr. D.K.J. Heylen

Abstract

The Virtual Storyteller is a story generation system based on emergent narrative. Emergent narrative is a method in story generation in which the characters all have their own goals, desires and motivations upon which they base their actions within the story. This way, the story is formed around the characters, instead of the characters adapting themselves to the story. While this does increase the believability, it makes it harder to generate an interesting story; the characters choose their actions based on what is best for them, instead of what makes a story interesting.

In this thesis, it is tried to remedy this, and make the stories generated by the Virtual Storyteller more interesting. This is done by following Freytag's dramatic structure, which describes how the conflicts in a story develop. Therefore, this thesis focuses on introducing and resolving these conflicts within the Virtual Storyteller, while trying not to impact the characters' autonomy significantly.

The conflicts are described by so called *plot threads*, that describe requirements the story world must meet in order to place two characters into conflict with each other. By making sure the world meets these requirements, the environment of the characters can be changed to instigate the conflict between them.

This conflict can then be resolved by giving the story's protagonist access to *slightly* better abilities than his antagonist, that appear to make him just a bit better or luckier, thus giving him enough of an advantage in the conflict to allow him to win it.

By controlling when new conflicts are introduced, and when the protagonist gains access to his better abilities, it becomes possible to direct enough to allow the action to rise, reach a climax, and then fall, and thus follow a dramatic structure.

Preface

Finally, after far too many months, I managed to finalize my thesis. The experience was ranging from frustrating (*"*sob* Why won't your fire that cannon, Guybrush?!"*) to really weird (*"Hmmz, technically you* do *win by blowing up your own ship."*), to just plain awesome (*"Both ships are destroyed, everybody loses! Yay!"*). Overall it was very enjoyable because, well, writing about wacky pirate hijinks just is.

A lot of gratitude goes out to Ivo and Mariët, for being understanding about my RSI-related problems, as well as their invaluable advice and guidance. Also, thanks go out to my parents, for their support and for not keelhauling me while my study was taking longer than anticipated. Furthermore, a "yay, beer!" goes out to my friends and my Atak-colleagues for giving me something to do in the periods I was not able to touch a computer. Finally, thanks go out to LucasArts, for their *Monkey Island*-series, that provided me with piratey inspiration.

Pjotter Tommassen.
February 24, 2009

Contents

Chapter 1

Introduction

The field of story generation concerns itself with the artificial, computer based creation of stories; i.e. instead of a human author writing the story, a computer generates (parts of) it. The generated story can be presented in numerous ways, e.g., the story in a video game, in writing, animation. By generating the story, instead of writing it directly, the story can for example change according to the player's interaction with it, be adjusted for the reader's preferences, or allow multiple viewings without becoming repetitive.

The Virtual Storyteller [Swartjes and Theune, 2008] is a story generation system that creates stories by means of emergent narrative. This is a method in story generation in which the actions taken by the characters in the story are motivated by their goals, plans and emotions, instead of by what actions facilitate an exciting story [Aylett, 1999]. One of the advantages of this method is that the characters behave in a believable way; because they are not being steered, the characters motivate the actions they take themselves, instead of them being forced to do something for the purpose of the story. This way, the outcome of the story is based on the actions of the characters, instead of being directly scripted by the author. There is also a downside to this advantage: characters do not choose their actions based on, for example, how suspenseful it will make the story, but based on how it will benefit the characters themselves, which does not always overlap. Therefore, it becomes a lot harder to introduce elements like conflict, suspense and excitement into the story. Since these elements are desired, if not required, in a good story, this thesis attempts to find a way to introduce them into emergent narrative in general, and the Virtual Storyteller in particular.

In order to add these elements to a story, it is investigated if the generated stories can be made to somehow adhere to a dramatic structure. This is a structure that describes the general flow of the conflicts in a story, and therefore the suspense and excitement in it. Because it describes the state of conflicts specifically, it makes sense to focus on them in this research.

This thesis tries to answer how and when conflicts can be introduced and resolved within the stories generated by the Virtual Storyteller, while trying to maintain most of the characters' autonomy. This is done by looking at how other similar projects handle dramatic structure and conflict, looking at what conflict means in the context of emergent narrative, and finally by designing and implementing a method that handles conflict in the Virtual Storyteller.

This thesis is laid out as follows: In chapter 2, it will be discussed what dramatic structure is, and what it adds to a story, focussing on Freytag's pyramid. Also, in the same chapter, an overview will be given on how other story generation projects deal with dramatic structure and conflicts, specifically noting elements that are used in this thesis. Finally, this chapter will describe how this leads to the main research question, 'How can we introduce the structure of Freytag's pyramid in emergent narrative while still maintaining the autonomy of the characters?', and the different subquestions that arrive out of it. Then, chapter 3 discusses the Virtual Storyteller, the story generation system that forms the specific context of this research, in more detail, after which chapter 4 describes a design that makes the stories generated by the Virtual Storyteller follow Freytag's pyramid. Then, in chapter 5, it will be discussed how these stories can be authored in the Virtual Storyteller, after which chapter 6 will present and analyze the two resulting stories. Following this, chapter 7 draws conclusions based on the results of this study. Finally, chapter 8 presents several options for future work based on this research.

Chapter 2

Dramatic Structure and Emergent Narrative

As mentioned in the introduction, the aim of this research is to introduce dramatic structure into emergent narrative. First, in section 2.1, the problems with emergent narrative with regards to controlling the story flow are discussed. Following, in section 2.2, dramatic structure itself will be discussed, i.e. what it is, why it is important, and different approaches to this structure in literature. Then, in section 2.3, the difficulties of applying this structure to emergent narrative will be discussed, as well methods used by other projects to address this problem. This leads to section 2.4, in which the specific problems and questions I want to address in this research will be discussed.

2.1 What is Emergent Narrative?

As mentioned in the introduction, emergent narrative is a method in story generation in which the characters choose their own actions, thus forming the story, instead of the characters being steered to facilitate the story [Aylett, 1999]. This is opposed to methods in which the characters are directly steered from 'above', i.e. the characters perform actions based on what the story requires of them, instead of what makes the most sense for the characters. The advantage of emergent narrative is that the flow of the story is not determined beforehand; the characters themselves determine the outcome of the story. This also makes it very suited for interactive stories; since the characters are autonomous, the player can theoretically take control of one, and influence the story the same way as the other characters can.

This autonomy is also emergent narrative's main disadvantage; because the author has limited influence on the story, it becomes a lot harder to introduce elements like conflict, suspense and excitement. The characters themselves do not spontaneously perform actions just to make the story more exciting; why wait for your rival pirate to show up when you can just pick up the treasure and leave? For the character to do so would reduce his believability. Also, forcing the pirate into waiting for his rival kind of misses the point behind emergent narrative, since the character did not choose his own action. Therefore, some way needs to be found to coerce, badger into, bribe, or even politely ask the

characters to help in creating an interesting story, while still maintaining most of their autonomy. In this thesis, it is tried to do this by having the characters, and therefore the story, follow a dramatic structure, as described in the next section.

2.2 What is Dramatic Structure?

Dramatic structure, or plot structure, refers to the parts in which a story can be divided. These parts are so generic that they can be found in any story, assuming it is of the genre the dramatic structure is based on; e.g., the *narratemes* identified by Propp [1968] can be found in any Russian folktale, but not necessarily in a modern horror story. Ensuring presence of these story parts is generally assumed to help greatly in crafting a complete story. The best known structure is the one introduced by Aristotle [1902], dividing the story into a specific beginning, middle and end, each with their own purpose within the story.

As has been stated before, in order to make the stories generated by emergent narrative somewhat more interesting, this research attempts to have the resulting stories follow a dramatic structure. In order to do so, one specific structure needs to be chosen to follow. This section discusses four structures, and concludes with choosing one of them.

2.2.1 Aristotle

Aristotle [1902] divided Greek drama's into three parts: the beginning, middle and the end. They are also known as setup, confrontation and resolution.

In the beginning, the conflict in the drama is set up; the characters and location are introduced, as well as some inciting incident that starts the conflict. Then, in the middle, the protagonist tries to deal with this conflict, usually resulting in more and more problems and ending in the climax, pretty much a situation that can not get any worse. Finally, in the end, the conflict is resolved in one way or the other, and the results of that resolution are described.

2.2.2 Freytag

Gustav Freytag [1900] analyzed the structure of Greek and Shakespearean drama's. He split a drama in five parts, also called Freytag's Pyramid. This structure is similar to Aristotle's three-act structure. First, there is the *exposition*, conforming to Aristotle's beginning; it introduces the characters, the location, and ends with the inciting incident. Following this is the *rising action*, introducing even more complications to the conflict. Diverging from Aristotle, this is followed by the *climax*, seen as a separate point by Freytag, instead of just as a part of a phase. In it, the reversal of fortune (i.e. going from the problematic situation in the second part to a more enjoyable situation for the protagonist) takes place. Then, during the *falling action*, the results of this reversal are shown, resulting in an even more enjoyable situation for the protagonist, somewhat conforming to Aristotle's end. Finally, the end result of the conflict is shown in the dénouement (e.g. "And they lived happily ever after").

2.2.3 Propp

Vladimir Propp [1968] analyzed a large number of Russian folktales. By breaking down these stories into the smallest narrative unit that describes some form of plot advancement, he concluded that there are 31 of these units, or narratemes. While not all of these narratemes are present in every story, they (nearly) always occur in the same sequence, and none of these stories diverge from them (i.e. nothing takes place in the stories that is not described by these narratemes). Some examples of these narratemes (also known as *Propp's functions*) are:

- A member of a family leaves home.
- Misfortune or lack is made known.
- Hero reacts to actions of future donor.
- False hero presents unfounded claims.

As can be seen, these narratemes are more specific than the parts in Aristotle's and Freytag's structures.

2.2.4 Campbell

Joseph Campbell [1973] explores the theory that most important myths from around the world that survived to this day follow the same basic structure, the so called monomyth. This structure consists of five stages; first, there is the *call to adventure*, in which the hero leaves for adventure. This is followed by the *road of trials*, where all sorts of trials are encountered by the hero that he has to overcome. After this, the hero *achieves the goal or "boon"*, usually resulting in important self-knowledge. Hereafter, the *hero returns to the ordinary world*, followed by the *application of the boon*, where the hero uses what he has learned to better the ordinary world.

A story based on Campbell's monomyth will nearly automatically follow Freytag's structure, as the five stages correspond roughly to Freytag's five parts. However, Campbell's structure only applies to a specific type of story; i.e. a story based on Campbell adheres to Freytag's structure, but not necessarily vice versa.

2.2.5 Conclusion

Of these four structures, Propp's and Campbell's might seem the easiest to implement at first sight; they describe (relatively) concrete events that should happen in every story, so just having these events take place in a story makes it follow one of these structures. However, this imposes a certain type of story, pending on the events described in the model (a Russian folktale, and an heroic journey respectively). Also, more importantly, it becomes a lot harder to let these events take place in the case of emergent narrative, as characters need to be rather restricted to follow these events, thus limiting their autonomy.

Freytag's and Aristotle's, however, are more abstract. Because they do not rigidly define what should happen in a story, it makes them harder to apply computationally, but also less limiting, and therefore more suited to emergent

narrative. Since Freytag's and Aristotle's structures are rather similar, I have chosen Freytag's due to personal preference; because it consists of five distinctively named parts, instead of three parts with rather generic names, it becomes easies to refer to each part individually within this thesis.

2.3 Dramatic Structure in Virtual Storytelling

One of the main problems with story generation systems (including interactive dramas), is to not only generate a sequence of events, but to actually make it an interesting story, i.e. give it some form of dramatic structure. This section will discuss how (and whether) this problem is approached by other systems, and what my research is trying to add.

2.3.1 Tale-Spin

One of the first story generation systems, Tale-Spin [Meehan, 1977], generates fables based on user input; before generation, the user describes the characters in the world and gives them some goals to achieve. Then, the agents controlling each character start making plans based on the character's goals and his knowledge of the world, after which each character starts following his plan, assuming it isn't interfered with. As with other emergent narrative systems, the story follows from the actions the characters take, and only from those actions; the system has no concept of the plot that is being created, and does not interfere with the characters. How interesting the plot is, is mostly dependent on the parameters given by the user; if the user gives the main character a complicated problem, the resulting story is usually more interesting. However, it is still possible for the tales to be 'mis-spun', meaning that something happens that prevents a consistent plot from occurring (e.g. the main character slips, falls into a river and dies, while he was actually looking for some food). When a tale is correctly 'spun', a rather nice fable is the result, though it only coincidentally follows any form of dramatic structure.

Lessons that can be learned from Tale-Spin are that the plot, and how interesting it is, is in large dependent on the human authored elements of the story, e.g. the characters, their goals, the environment. Also, the generated fables seem mostly consistent due to the rather small domain and story length; because the characters can perform relatively few actions, that are also quite specific to the story, the chances of them performing an action that does not contribute to the story are rather slim.

2.3.2 TAILOR

A different system is TAILOR [Smith and Witten, 1987]. Similarly to Tale-Spin, it keeps track of a virtual world in which characters try to achieve their goals. However, it differs in that one character can be an antagonist, trying to sabotage the plans of the protagonist. The system has an evaluation function that calculates how favorable each potential storyworld state is for the protagonist. The protagonist obviously tries to reach the states most favorable to him, while the antagonist tries to achieve the lower-valued states. Since the plans of the agents are usually directly influenced by the other, they have to take each other

into account when creating a plan. TAILOR does this by means of the minimax [Wooldridge, 2002] algorithm; every possible action in the plan will not only be evaluated by the direct result of the action, but also by how much it will help the other character. In order to ensure that the antagonist and protagonist interact (thus making the conflict actually visible to the reader), the system sometimes allows the antagonist to take extra actions, in effect making the protagonist skip turns.

Due to the antagonist, TAILOR introduces conflict into every story, which has the potential to make the story more interesting, and is also required if one wants to follow Aristotle's or Freytag's dramatic structure. However, the antagonist has no real motivation to hinder the protagonist, making him less believable. The concept of making characters skip turns to have them meet is rather interesting, and is partially used within this study to make sure that the antagonist does not directly win while the protagonist is getting in all sorts of hijinks (see section 4.3.3).

2.3.3 FearNot!

FearNot! [Aylett et al., 2006] is an emergent narrative system that uses emotionally driven characters. However, unlike most other story telling systems, its main purpose isn't to make the story entertaining, but for it to be educational. The characters in the world are a bully and a victim, with the bully having the goal to make himself feel better by annoying the victim. When the simulation in FearNot! is running, it plays an episode in which the victim gets bullied, after which the player has the chance to tell the victim how to deal with it. The victim's behavior is then changed to take the player's reaction into account, after which the episode is restarted.

This system can be seen as a very advanced version of a system like Tale-Spin. The story takes place in a number of episodes. A human author describes the state of the characters and the world at the beginning of each episode. When the Story Facilitator [Figueiredo et al., 2006] (a subsystem similar to the Virtual Storyteller's Plot Agent, as described in section 3.2.3) starts an episode, it brings the characters and world in the state described in that episode definition, after which the characters are controlled by individual agents without outside influence. The characters act upon what is best for them, instead of what is best for the story, meaning that the only way to ensure an interesting story is by making sure the starting situation of each episode leads to one. In the case of FearNot!, the bully-character is given the goal to bully the victim-character, which he will attempt to do, thus resulting in a story about bullying behavior. As discussed in section 4.1, a method similar to this is used within this study to set up the story's conflict.

2.3.4 Façade

A completely different approach is used by the Oz-Project [Mateas, 1999], and its follow-up, Façade [Mateas and Stern, 2003]. Both projects concern interactive story systems, in which the characters not only determine what to do, but are also steered by a drama manager. In the case of the Oz-Project, the drama manager has a library of predefined plot points, which are important moments within a story. Between plot points, the character agents (and the user) are

pretty much allowed to do whatever they want. The drama manager tries to detect a possible transition to another plot point, after which it will steer it towards that plot point.

With Façade, the drama manager has a larger preprogrammed library of *story beats*, very small sequences that can occur within a story. Once, by means of the player's actions, preconditions of a certain story beat are met, it will be played, controlling the behavior of the characters. The frequency with which the beats play is a lot higher than that of Oz's plot points, thus steering the story even more.

By means of these plot points and story beats, it becomes possible to introduce a form of dramatic structure; conflicts can be introduced, their build-up can be ensured, and a resolution to them can be built-in. However, the points and beats severely confine the characters' autonomy, and limit the number of stories that can be generated, due to all potential flows of the story already being scripted.

2.3.5 Fabulator

This latter disadvantage, of the story flows being scripted, is somewhat lessened in the Fabulator project [Barros and Musse, 2005]. It is an interactive story system in which, instead of scripting pretty much every state the world can be in, only three states are described, corresponding to Aristotle's three-act structure; each state describes how the world should be at the end of the corresponding act. The characters are all controlled by a single planner which steers the characters towards the state corresponding to the current act. Once that state has been reached, the end of that act has been achieved, and the next one can be started, meaning that the planner will now try to steer the characters towards the state describing the end of the next act.

Because of the three defined states, it becomes rather predictable what will happen in the story in general, but the details of how these states will be reached are not scripted at all. An approach similar to this is used in this study to determine how far a conflict is from being resolved, as discussed in section 4.2.1.

2.3.6 I-Storytelling

Somewhere between the emergent, character controlled narratives of Tale-Spin, Tailor and FearNot!, and the scripted, drama manger controlled narratives of the Oz-Project, Façade and Fabulator lies I-Storytelling. In it, different characters act in a virtual world upon which the player has some limited influence; he can e.g. move objects, or give advice to characters, but he isn't considered a full character himself. The character's behavior is based on predefined Hierarchical Task Networks (HTN's), a tree-like structure that describes the actions a character can perform in every situation in order to achieve his goals.

While this means that the characters are not steered by any form of drama manager, their behavior, and therefore the situations that occur in the story, are limited by their HTN; it prescribes all possible actions a character can take, so the resulting stories are still scripted, just from the perspective of the characters instead of the entire story.

2.3.7 Conclusion

What I personally think is missing in this spectrum of storytelling systems is some merger between the approaches; i.e. the characters should maintain their autonomy, but it should still be possible to direct the entire story in some sort of dramatic structure, making it possible to actually be exciting. The problem with this in emergent narrative is that the characters usually do not have a sense of the story; their view of the world is based on their own beliefs, desires and intentions, and not on e.g. whether they are approaching the climax of their conflict. Oz-Project and Façade solve this by means of a drama manager, that keeps track of the story, and corrects characters if they behave in a way that does not contribute to it. However, it keeps the characters on too tight a leash; they cannot leave the boundaries set by the drama manager. While this does give the author more influence on the resulting story, it is not really emergent narrative.

However, I think it should still be possible to direct the characters into the direction of a coherent story without steering them so forcefully. FearNot!, as well as previous Virtual Storyteller work, shows that it is already possible to have episodes of an emergent story go in a way that was intended by the author without steering it after the setup. Therefore, it makes sense that this might also be possible on a larger scale, i.e. per story instead of per episode.

2.4 Research

As discussed in section 2.2.5, Freytag's pyramid was deemed a dramatic structure possible to apply to stories generated by emergent narrative. This section will first discuss the pyramid in greater detail, which leads to the research questions that will then be discussed.

2.4.1 Freytag's pyramid

In section 2.2.5, it was chosen to apply Freytag's model of dramatic structure to the Virtual Storyteller. Therefore, it is explained in more detail in this section.

Freytag [Freytag, 1900] analyzed the structure of Greek and Shakespearean drama's. He concluded that a drama could be split into five parts, also called Freytag's Pyramid (figure 2.1). The individual parts are discussed below.

Exposition During the exposition, the main characters and the setting of the story are introduced to the audience. During the exposition, the plot does not necessarily advance; the exposition can be done by means of a narrator that is not part of the story per se. Also, some action might happen, but it is mainly to introduce the main characters instead of having anything to do with the primary conflict of the story.

At the end of the exposition the inciting moment takes place, the event that launches the primary conflict of the story. This inciting moment might be the cause of the conflict, or might give an explanation as to why the conflict suddenly becomes of interest, i.e. why the telling of the story starts at that moment, instead of at a previous time in the story world.

Examples of inciting moments are:

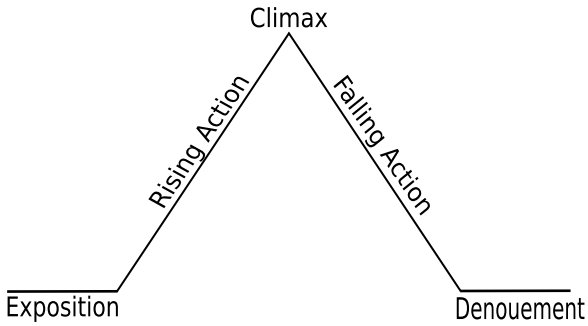

Figure 2.1: *Freytag's Pyramid*

- The protagonist discovers the map to a hidden treasure.
- The pirate princess the protagonist fell in love with is kidnapped by an evil pirate.
- A cursed ghost ship attacks the porttown the protagonist is at.
- The protagonist is given a sign that he has a debt to pay.
- The protagonist sees the ghost of her father pass by, causing her to swear revenge on his killer.
- The protagonist accidentally resurrects an evil pirate.

Rising Action During the rising action it is attempted to maintain the suspense in the story, by introducing secondary conflicts that form obstacles for the protagonist. These conflicts can either be resolved during the rising action itself (usually if they are not directly related to the primary conflict), or be resolved during the falling action if they are somehow related.

Examples of secondary conflicts during the rising action are:

- A mutiny takes place.
- A series of tasks needs to be performed before the protagonist gets all the pieces of the map.
- The ship strands on an uninhabited island.
- A group of blood-thirsty monkeys takes the protagonist prisoner.
- The East Indian Trading Company starts to annoy the pirates.

Note that some of these secondary conflicts can just as well be primary conflicts; the differences and similarities between them are addressed in section 4.3.1.

This phase is usually the longest in a story.

Climax The climax is the turning point of the story; it indicates the moment from where things finally start to go well for the protagonist if the story has a happy ending, or when things start to get seriously bad in the case of a tragedy. Since the pirate fiction I am most familiar with features primarily happy endings, those will be focused on mainly.

Falling Action During the falling action, the primary conflict, as well as any unresolved secondary conflicts, are unraveled. It deals directly with the results of the actions taken during the climax and shows e.g. the antagonist getting beat, the protagonist capturing the treasure, finally managing to free the pirate princess, etc. Resolving all these conflicts usually happens a lot faster than introducing them.

Dénouement The dénouement happens when it becomes obvious that the primary conflict has been resolved; i.e. the protagonist has won, and there is nothing the antagonist can do to change it. It usually shows the results of the conflict's outcome, e.g. the pirates celebrating their victory with rum, the protagonist marrying the rescued princess.

2.4.2 Questions

The main question in this research is 'How can we introduce the structure of Freytag's pyramid in emergent narrative while still maintaining the autonomy of the characters?' Since this question is rather broad, it is split into smaller subquestions.

As was shown in section 2.4.1, Freytag's pyramid concerns itself largely with the conflicts in a story, i.e. when they are introduced and when they are resolved. Therefore, the research questions focuses on conflicts as well.

Single Conflict Since it is difficult enough to deal with just a single conflict, this will be addressed first. Subquestions with regards to a single conflict are:

1. How can a conflict be introduced? If characters have no reason to dislike each other, or have some form of conflicting goals, no conflict will occur. Therefore, this must be introduced in some way.

2. How can the resolution of conflicts be facilitated? A story will feel rather unfinished if there are still unresolved conflicts once it ends. However, the characters might feel like they have better things to do, or just not be capable of resolving a conflict. Some method needs to be devised to compel the characters to resolve their conflict somehow, without directly forcing them to do so.

Larger Structure Once it becomes possible to actually have a single conflict, it becomes possible to have more of them and place the conflicts in a structure like Freytag's. Subquestions concerning this are:

3. When should conflicts be introduced? Besides the obvious primary conflict that should be introduced at the start of the story, the secondary conflicts in the rising action should also be introduced at some time. When is the right time to 'make the story even more exciting'?

4. What makes a conflict a secondary conflict? Besides 'not being the primary conflict', are there any other criteria that distinguish a secondary conflict from the primary, or can a secondary conflict just as well be a primary conflict in a different context?

5. When is the right time to resolve the conflicts? In Freytag's pyramid, resolution of the primary conflict, as well as most secondary conflicts, takes place after the climax, during the falling action. How do we determine when this climax should take place?

6. When does a story end?

The questions are addressed by first looking at what other projects have done with regards to these specific questions. Then, a look is taken at what happens in 'real' stories, i.e. the ones we try to generate with the Virtual Storyteller. Inspired by this, a design is made of a sub-system in the Virtual Storyteller that makes the generated stories follow Freytag's structure. This system is subsequently implemented and tested.

Chapter 3

The Virtual Storyteller

This chapter describes the Virtual Storyteller project, the context in which this research takes place. First, in section 3.1, a short overview of the project is given. Then, in section 3.2, it is described how the Virtual Storyteller generated a story before this research took place, so that this can be expanded upon in chapter 4, which discusses how dramatic structure can be added to these stories.

3.1 Overview

The Virtual Storyteller is a project carried out at the University of Twente. Its aim is to generate stories, which can then be told to the user [Swartjes and Theune, 2008].

The Virtual Storyteller generates stories by means of emergent narrative. It started out as a project for a master thesis [Faas, 2002], in which a system was developed that is able to generate some simple stories and tell them by means of the Microsoft Agent software [Microsoft Corporation, 2003]. This project was quickly expanded upon in various directions, among them better natural language generation [Theune et al., 2007], emotional characters [Theune et al., 2004] and more interesting plot generation by means of episodes and believable characters [Rensen, 2004]. The plot generation has been expanded

Figure 3.1: An overview of the agents in the Virtual Storyteller.

upon in Swartjes [2006], in which the architecture of the Virtual Storyteller was redesigned.

Currently, the Virtual Storyteller is a framework that generates stories by means of simulation: virtual characters exist in a virtual environment. They have goals they try to achieve, by means of actions that influence the world. The actions those characters choose then make up (most of) the story. Based on these actions, the Virtual Storyteller generates the fabula, a graph that contains all actions and explains why the characters performed them. Once the simulation is done, the fabula can be read by a computer-based narrator in order to present the story to a user, for example in the form of natural language, a comic strip or a movie. The story generation process is discussed in more detail in the next section.

The Virtual Storyteller framework is used to create an application capable of generating pirate themed stories, called PLUNDER. Work is being done on components like a new emotional model, and a reimplementation of the narrator.

3.2 Story Generation

In the Virtual Storyteller stories are generated by means of simulation. Each character is controlled by a separate artificial intelligence. These characters exist in a (human authored) virtual story world. They are capable of perceiving things that happen in this world, and performing actions to manipulate it. The characters are given story goals, which their AI tries to achieve by means of the aforementioned actions. The actions performed by the characters then form the content of the generated story.

All this is done with a multi agent system. The three different types of agents, shown in a diagram in figure 3.1 are described below:

- **Character Agent.** Character agents form the AI of the characters in the world, thus controlling their actions and responding to their perceptions.
- **World Agent.** The world agent keeps track of the state of the virtual story world, and determines the result of the characters' actions.
- **Plot Agent.** The other two agent-types all interact through the plot agent. Its main responsibility is to keep the plot interesting and consistent.

Below, the agents will be discussed in more detail, followed by an overview of how this all fits together.

3.2.1 World Agent

As mentioned above, the world agent keeps track of the state of the story world. It does this by means of a knowledge base, containing facts about the world. Facts might be that there exists a treasure chest, that there is gold in the chest, and that the chest is at a beach. Besides information about the world itself, the knowledge base can also contain story related facts, like whether a character is the protagonist or not; these type of facts are barely used at the moment, but will be used within this research. This story related knowledge might be seen as something the plot agent should keep track of instead of the world agent, but at the moment it is not treated as anything different than the 'normal' world knowledge.

The facts are described by a collection of RDF-triples [RDF]. For example, the treasure chest mentioned above can be represented as such:

```
(oTreasureChest1 a TreasureChest)
(oTreasureChest1 contains oGold1)
(oTreasureChest1 supportedBy oBeach1)
```

The above tells the world that the object `oTreasureChest1` is of the class `TreasureChest`, that it contains the object `oGold1` and that it is located at `oBeach1`.

As well as these facts, the world agent has access to an ontology in OWL [OWL]. This ontology describes the relations between different classes in the world, for example, stating that the class `TreasureChest` is a subclass to `Container`, thus allowing other objects to be put inside of it. This allows reasoning about the world.

Besides just knowing what the world looks like, the world agent also facilitates changes to the world, like the results of actions performed by characters. These actions are described by means of operators. An operator is a thing that changes the world state. There are multiple types of operators, of which actions are one. The different types are discussed a bit further below.

An operator is described in a schema that consists of a number of elements, the most important of which (in the context of this thesis) are described below:

- **Name.**[1] The name of the operator, e.g., `PickUp` or `ShootCannonAtShip`.
- **Arguments.** The arguments of the operator describe the variables that need to be filled in into the preconditions and effects (both are described below), in order for the operator to be performed, e.g., the `Agens`-argument states who is performing the operator, and the `Target`-argument states who or what is targeted by it. Most of the arguments are optional (the exception is that actions need an `Agens`-argument), and there is no strictly defined list of possible arguments. Also, not every variable in the operator needs to be stated as an argument, just the ones that change how the operator is performed; e.g., when picking something up, the thing that is being picked up would be an argument (the `Patiens`), while the location of the object being picked up would not be one, since this cannot be changed by performing the action differently (e.g., one cannot suddenly pretend

[1] Actually called 'Type' in the implementation. In order to avoid confusion with the different types of operators, it is called 'Name' in this thesis.

that the treasure is on the ship while picking it up, while it is actually on the beach).

- **Duration.** The number of story rounds this operator takes to perform. If an operator does not have a duration, it takes no time to perform; its effects are instantanious. Story rounds are explained in more detail in section 3.2.3.
- **Preconditions.** The preconditions describe a number of conditions that need to be true in order for the operator to be performed. These conditions will usually be facts that need to be either true or false in the world knowledge base (e.g., the treasure chest is open, but not empty), or rules concerning reasoning about the world (e.g., the treasure chest is some sort of container). More types of conditions might be added in the future. The preconditions can also fill in variables that were not filled in by arguments; e.g., it can be stated that `Agens` and `Patiens` (both arguments) need to be at the same `Location`, where `Location` will be filled in depending on the actual location of the `Agens` and `Patiens`. (Note that if they are not at the same location, the variable cannot be filled in, and the preconditions cannot be met.)
- **Effects.** The effects describe the result performing this operator will have on the world. It is described by means of facts that are either added to or removed from the world knowledge base.

An example operator, an action that picks stuff up, is shown below (paraphrased for readability):

Name:	`PickUp`
Arguments:	`Agens` and `Patiens`
Duration:	1 round
Preconditions:	`fact({Agens supportedBy Location})` is true `fact({Patiens supportedBy Location})` is true `rule(Patiens typeOrSubType carryable)` is true
Effects:	`fact({Patiens heldBy Agens})` becomes true `fact({Patiens supportedBy Location})` becomes false

In short, it requires the character doing the picking up, as well as the object to be picked up, to be in the same location, and the object to be picked up to actually be something that can be picked up. If the operator can be performed, it will remove the picked up object from its old location, and put it in the hands of the character performing the operator.

The plot agent can request an operator to be performed by the world agent and ask a character agent for an action to perform, which it then passes to the world agent. Character agents can additionally pass a request to the plot agent to perform an out-of-character operator (i.e., a framing operator or an event, discussed below), which the plot agent will then pass on to the world agent. The world agent then verifies whether the operator's preconditions are true, in order to maintain consistency within the world, in which case it can be performed. Otherwise, the request is denied. If the request can be performed, the world

agent confirms this to the plot agent. The effects are carried out, and the plot agent sends the corresponding changes in the world to the character agents.

At the moment, there are three types of operators, though more are added within this research. These operators can be divided in two groups, the in-character operators, and the out-of-character operators. In-character operators are operators that are actually performed by the character, i.e., of which the characters is the direct cause, in story. The most obvious one is the action: when a character picks something up, it is definitely the character that does it.

The out-of-character operators, conversely, cannot be directly associated with a character; while the outcome of such an operator may certainly benefit a specific character (and the character's agent can very well request that operator to be performed), in story the character will not be seen as the direct cause. The most obvious example is an event; if a gust of wind blows the character's ship towards the island with the treasure, he is probably very happy about it, but he did not directly cause the gust of wind. Since no performing character is associated with an out-of-character operator, these operators can also be directly performed by the plot agent, if there would be any cause for this (which there will be, as discussed in section 4.1).

The current three types of operators are described below:

- **Action.** An action operator is an in-character operator that describes an action a character can take when trying to fulfill his goals. Examples include picking things up (requiring that the object is in the vicinity of the character, and having the effect that he has it in his possesion) and drinking rum (requiring the character to have a bottle of rum, with the effect that the bottle is now empty and the character drunk).

- **Event.** An event operator is similar to an action in that it describes something taking place. However, an event is an out-of-character operator, and therefore not intended by a character, at least from a story point of view. Examples are a sudden gust of wind blowing the ship towards the island, a cave-in or a character suddenly stumbling. As discussed above, even though these events are not caused by characters, they can be planned in; if it would be very useful for a character to suddenly get blown to another island, his planner can request the event to the plot agent, hoping it takes place.

- **Framing Operator**. A framing operator is an out-of-character operator that introduces new elements to the world, and pretends that they have always been there [Swartjes et al., 2008]. This way, it is not necessary to fill in the entire world beforehand; if a bottle of rum is needed for some reason, it can be introduced by a framing operator, instead of requiring it to be defined at the start of the simulation. A character can request a framing operator to be performed if it fits his goals. Note that in the context of the story nothing really happens when a framing operator is used; the introduced bottle of rum didn't just appear, but it has always been there and just hasn't been mentioned yet.

3.2.2 Character Agent

A character agent controls a character in the story[2]. It keeps track of the character's beliefs about the world and the goals it wants to achieve in it. The agent is the part of the character that makes it 'alive'. Without the agent, the character would be treated as just about any other entity in the story world, and would not be very different from, say, a tree.

At the time of this writing, the characters don't have any emotions, though a model for this is being developed, and will be part of the character agent.

The character agent receives perceptions from the plot agent. These perceptions are usually the result of performed operators. By means of these perceptions, the character's beliefs are changed, similar to how the operators themselves change the world.

A character agent has goals, that are described in goal-schemas. A goal-schema contains the following elements:

- **Name.** The name of the goal.
- **Preconditions.** A set of conditions that need to be met in order for a character to adopt this goal, similar to the preconditions of an operator. This is optional. Goal adoption is discussed further below.
- **Succes Conditions.** A set of conditions that indicate when a goal has been achieved.
- **Failure Conditions.** A set of conditions that indicate when a goal cannot be achieved anymore. This is also optional.

In order to achieve its goals, the character agent builds a plan by means of a partial order planner [Kruizinga, 2007]. Such a plan consists of operators that can either be performed by the character (at least, according to the character's beliefs), in case of actions, or that the character requests to take place, as is the case with events and framing operators.

To plan for actions other characters can take, a character can us an expectation-schema in his plans. An expectation-schema is a schema very similar to an operator that describes expectations characters might have. These are used by characters for things that they cannot directly influence, like the behavior of other characters, but for which some model is required in order to make their plans. Unlike operators, an expectation schema can not be executed; its effects can be used to fill in preconditions in a character's plan, but he has no real influence on whether it works or not. For example, to facilitate characters buying a ship, an expectation might exist that has the precondition that money is given to a used ship-salesman, and the effect that the character that gave the money now has a ship; the character has no real influence on whether the salesman will give him the ship afterwards, but he does expect it, allowing him to plan having a ship.

While an expectation-schema is, strictly speaking, not an operator (it does not operate on anything), it is so similar that it is treated as one in the rest of this thesis.

[2]Theoretically, a character agent can control multiple characters. While it might be useful to have a single agent control an entire horde of blood-thirsty monkeys, it is assumed that a character agent controls just one character in order to keep things readable.

Once every round (discussed in the following section), the plot agent requests an action from every character agent. Every character agent then sends the first action in its current plan to the plot agent, which then requests the world agent to perform it, as outlined in the previous section. If all goes well, the character agent then receives perceptions of the operator's results. If the first operator in a character's plan is not an action, this is because some preconditions of the first action aren't met yet. In that case, the first operator will usually be an event or a framing operator, the execution of which is then requested to the plot agent (which sends it to the world agent). If the character agent receives some new perceptions that enable the first action, usually caused by the operator it just sent, it will send that action to the plot agent. If the first operator in a character agent's plan is an expectation, nothing happens; the character is waiting for that expectation to come true.

As well as sending the operators, the characters also send parts of the fabula graph that explain the reason they chose that operator (usually to fulfill their goal).

The character agent has two ways of adopting goals to pursue. The first is that they take on a goal suggested by the plot agent. Characters can theoretically reject these goals if they do not mesh with their values, beliefs, personality, conflict with, or have a lower priority than, their current goal, etc., but at the time this thesis was written, there was no mechanism yet that could determine whether the goal should be accepted or not, thus making it difficult to take this into account. While this study does touch upon the possibility of suggested goals being rejected, no concrete solution is given to ensure that a goal will be accepted by a character.

The second method of adopting a goal is that characters can take on a goal if they have nothing better to do. At the moment, a character can choose any goal that is possible to complete, not taking any personality into account (since characters do not have any yet). Due to this, and the mechanism not yet being complete, this is also mostly ignored within this study.

Because the goal adoption mechanisms described above are not finished yet, they are ignored within most of this study. In chapter 7, it is discussed how these systems can theoretically impact the techniques introduced in this thesis.

3.2.3 Plot Agent

The plot agent mediates between the characters and the world agent, meaning all communication between the latter two goes through the plot agent.

At the moment, it primarily receives requests from characters to perform an operator. It passes these on to the world agent. The world agent can then either refuse or allow the operators (pending on their preconditions), and sends this answer to the plot agent. The plot agent passes the results of the operators (if allowed) on to all the characters; if something actually happend in the world (as with events and actions), the results are passed on as perceptions. If nothing 'really' happened, as is the case with framing operators, the results are passed on as new settings. While it can theoretically refuse operators, or distort (or even withhold) perceptions, it has as of yet no mechanism to decide when this would be useful, so this is not used at the moment.

Besides passing on information, the plot agent also manages the story rounds and the fabula.

The story rounds are pretty much what the name implies: rounds of the story. The simulation is performed in discrete rounds; the plot agent tells all other agents that a new round has started, requests an operator from every character agent, and awaits the results of these operators from the world agent, after which they are passed on to the characters, and a new round can start.

The fabula [Swartjes, 2006] is a graph describing what takes place in the story; all operators that were applied, the reason an operator is chosen, the beliefs of a character, the motivations behind their beliefs, etc. This fabula is built during the simulation. Though every agent can send additions to the fabula (e.g. the reason a character tries to perform an operator), the plot agent keeps track of the entire graph. Once the simulation is done, the narrator can present this graph into a form that is understandable to the end user.

3.2.4 Example Run

Now that all the agents and their responsibilities have been discussed, a view of how they fit together is given, by describing how a story is generated.

At the beginning of the story, the story setting is loaded. This not only contains the story world facts used by the world agent, but also the characters and their goals, as well as the operators they can perform. The plot agent then informs the character agents about the character they are controlling, after which the first round of the simulation can start.

During every round, the following happens:

1. The character agents make a plan to fulfill their goals.
2. The plot agent requests a first action in every character agent's plan.
3. The character agents pass on the first operator of their plans (not necessarily an action) to the plot agent.
4. The character also sends a part of the fabula describing the reason they requested this operator.
5. The plot agent might disallow some of the operators (though this doesn't happen at the moment). If not, it passes the operators on to the world agent.
6. The world agent verifies the preconditions of every operator it receives. If an operator can be performed, it tells this to the plot agent. If not, a refusal is sent.
7. The plot agent sends the effects of the operators to the character agents as new perceptions and/or settings they are receiving. The fabula is updated with the reason an operator occurred, as well as the effects and perceptions.
8. If the character agent didn't send an action, it keeps on sending the first operator in its (updated by the received settings) plan, i.e. repeating steps 3 to 7, until it has finally sent an action.
9. Once all character agents have sent their action (or couldn't send them because they can't make a plan), the round ends and a new one begins.

These rounds continue indefinitely, meaning that the story doesn't end; this problem is addressed in this research. It can be stopped manually though, after which the fabula graph can be saved and presented by a narrator.

Chapter 4

Conflict Management

As shown in section 2.4, the dramatic structure of Freytag's pyramid concerns the development of conflicts within a story. Therefore, this chapter discusses how conflicts can be managed within the Virtual Storyteller, introducing dramatic structure into the story that way. First, in section 4.1, it will be discussed how the primary conflict of a story can be introduced, after which section 4.2 describes how it can be resolved. Then, in section 4.3, the story is made more exciting by adding additional secondary conflicts. Following, section 4.4 discusses how the story should end. Finally, section 4.5 summarizes these sections.

Most concepts introduced in these sections are illustrated by means of examples. The main characters in these examples are all based on the Monkey Island series of adventure games, created by LucasArts.

4.1 Introducing Conflict

The first research questions is: 'How can conflict be introduced?'. If characters have no reason to dislike each other, or have some form of conflicting goals, no conflict will occur. Therefore, this must be introduced in some way.

This section will first discuss what conflict entails, and how it can be represented. Then, it will briefly recapitulate what other projects have done with regards to this question, after which it will be discussed how, by introducing the *plot thread*-device, it becomes possible for the Virtual Storyteller to introduce conflicts.

4.1.1 What is a conflict?

When wanting to introduce conflict, it is logical to first take a look at what conflict entails. Conflict can, in essence, be described as the protagonist's struggle against an opposing force, whether this is an antagonist, the environment, himself, society, or something else entirely. Because emergent narrative is driven by characters, this research focuses mostly on conflicts against an antagonistic character.

The conflict is most likely caused by the character having some values, opinions or goals that are in opposition with the antagonist's. Because the characters in the Virtual Storyteller do not hold any values or opinions yet, goal-based

conflicts will be the focus of this research.

Goals usually conflict because they are incompatible in some way. For example, assuming pirates do not split treasures, two characters wanting the same treasure have conflicting goals. Likewise, the blood-thirsty monkey's goal to feed on a pirate is incompatible with the pirate's goal to stay alive, and the hero's goal of marrying the princess is opposed by the goal of pirates to kidnap her.

What can be seen from these example conflicts is that the goals do not oppose each other because they are the direct opposite, but because achieving one goal makes it a lot harder, if not impossible, to achieve the other goal; LeChuck does not explicitly want Guybrush to not have the treasure, it's just that he cannot have the treasure while Guybrush has it too.

In a preliminary study it was tried to computationally find out whether two given goals are in conflict with each other; once it is known that two goals conflict with each other, they can be assigned to different characters, thus giving them opposing goals. This study was limited to goals that are mutually exclusive, i.e. achieving one goal makes it impossible to achieve the other one, because making one goal 'impossible' to achieve is easier to define than just making it 'harder' to achieve. In this case, a goal is defined as impossible if the character having it can not find a plan to fulfill it.

Following this definition, two goals can be seen as mutually exclusive if none of the involved characters can find a plan to fulfill both of them at once. This can easily be checked with the following scheme:

- Make sure each goal is possible to achieve individually.
- Give both goals as a goal to one of the characters' planners.
- Allow that planner to use actions performed by the other character (instead of just actions of the character that planner is assigned to) as steps for its plan; this can be seen as the other character fully cooperating to achieve both goals.
- Try to find a plan. If no plan can be found, it can be assumed that it is impossible for one character to achieve one goal while the other has fulfilled the other goal, even when the characters are cooperating with each other.

Unfortunately, not finding a plan can take quite a while, in some cases even taking hours, making this approach rather impractical. Also, just using mutually exclusive goals in stories is too limiting; while this approach will detect LeChuck and Guybrush wanting the same treasure as having conflicting goals, it will not detect Murray stealing Guybrush' ship as opposing; Guybrush might be able to build a new ship, allowing him to still find the treasure, even though the loss of his ship was rather annoying and hampered him immensely. Therefore, it seems that a better approach is to manually indicate that two goals are in conflict with each other.

4.1.2 Conflict in Other Projects

One of the first storytelling projects that introduced conflict with an antagonist was TAILOR [Smith and Witten, 1987]. The conflict in the story results from

the antagonist planning directly against the protagonist; where the protagonist chooses an action that evaluated as the most beneficial to him, based on how much it helps him in achieving his goal, the antagonist chooses the action evaluated as the most detrimental to the protagonist.

The problem with this method is that the conflict is not really introduced, from a story point of view; there is no reason given for the antagonist's attitude, and it does not make that much sense to just try and annoy another person without gaining something from it, reducing the character's believability.

Another project heavily based on conflict is FearNot! [Aylett et al., 2006]. In it, two characters, the bully and the victim, come in conflict with each other. This conflict is the result of the emotional model of the characters; the bully dislikes the victim, feels stronger than the victim, and therefore starts to pester him. The victim does not like this, and conflict ensues.

Since the characters in the Virtual Storyteller do not have an emotional model (yet), directly applying FearNot!'s approach of having characters dislike each other is impractical. Also, pirate-based conflicts are not necessarily driven by the emotions pirates have regarding each other (two pirates do not have to dislike each other to want the same treasure), and these kind of conflicts are not covered with this method. However, one of the aspects of this approach relevant to this research is that the conflict, as played out in the story, follows from the setting at the start of the story; when the story is started, the bully's dislike of the victim is manually assigned. Then, during the run of the story, the bully starts annoying the victim, without being steered by any external factor. This can also be applied to the Virtual Storyteller; by manually setting some specific attributes of the characters at the beginning of the story, it should be possible for conflict to ensue.

4.1.3 Design of Conflict Introduction

In this research, we are trying to cause conflict between characters by giving them a reason for it at the beginning of a story, similarly to FearNot!. However, the reason for conflict in the stories generated by FearNot! is that one character dislikes the other. While this is fine in a system helping children deal with bullying, it does not really work with stories about pirates; while it is certainly possible to write a pirate-themed story concerning the feelings the main characters have towards each other, most stories are more in the vein of the *Pirates of the Carribean* movies, or the *Monkey Island* games, were conflicts are about treasure and rum. Therefore, a method is devised to introduce these kind of conflicts.

This section will discuss the design of a system that introduces these conflicts in the Virtual Storyteller. First, the separate components used to facilitate this are discussed, followed by a description of how they fit together.

Below are the components that this research uses in the Virtual Storyteller to introduce conflict. The new components are discussed in more detail later on.

Since the conflicts are between characters, the character agents, introduced in section 3.2, play an important part in the conflict. However, since we want to control the characters as little as possible, the modifications to the character agents are kept to a minimum; the main modifications are made to the plot agent, also introduced in section 3.2, instead. This is done by adding two new

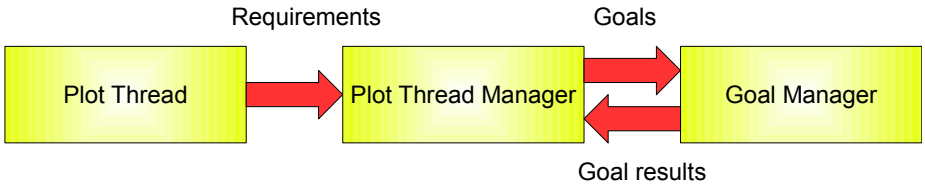

Figure 4.1: An overview of the new elements added to the plot agent to introduce conflict.

parts to the plot agent, the goal manager and the plot thread manager. The goal manager allows the plot agent to have goals and try to fulfill them, similarly to the character agents. The plot thread manager keeps track of introduced plot threads, a structure that is used to introduce new elements, like conflict, to the plot. The plot thread, goal manager and thread manager are discussed in more detail below.

Plot Thread The plot thread is a structure that describes a partial state the story world should be in in order for something, e.g. a conflict, to occur. While its usage in this thesis is limited to conflicts, it can potentially also be used to describe different, currently undefined, plot elements, like scenes, quests, etc.

A plot thread consists of the following parts:

- **Name** The name of the plot thread.
- **Requirements** Requirements regarding the story world that describe the partial world state. Will be explained in more detail in this section.
- **Type** The type of the plot thread. As discussed in section 4.3.2, this can indicate whether the plot thread describes a primary or secondary conflict.
- **Resolution-goals** Describes when the conflict stated in this plot thread is resolved, as discussed in section 4.2.1.

The requirements can be anything that the world agent keeps track of; the existence of objects, properties of existing objects, characters, character goals, loose facts, etc. By grouping them together in a single plot thread, it becomes possible to state multiple related requirements at once. Fulfilling these requirements can then introduce coherent new parts to the plot. For example, in a plot thread describing a treasure-related conflict, the thread can require a treasure somewhere on an island, as well as requiring two different characters to have the goal of finding said treasure.

The requirements regarding the world (with the exception of characters and their goals) are stated as a collection of facts that need to be true. When starting a plot thread, making these facts true, if they were not already, is given as a goal to the goal manager, which then tries to enable them, thus fulfilling the requirements, as discussed below.

The characters and their goals are treated somewhat differently. The plot thread describes the characters by means of some conditions that need to be true about a character, e.g., the character must be a pirate, and the protagonist. A goal can also contain a set of conditions that need to be true in order to have

the goal accepted by a character (e.g., the character must be a greedy pirate). If there already is a character that satisfies all these conditions, or is capable of satisfying them by means of framing operators, that character will be used. If not, a new character that does fulfill these conditions will be introduced. The character, whether already in the story, or newly introduced, is then offered the goal that the plot thread prescribes for him. The character can then either accept or refuse the goal (the goal might conflict with the character's values or something). If the goal is refused, it is suggested again to a different character that matches the required conditions. In the rest of the text, it is assumed that they will always accept a goal.

Note that it is possible that a plot thread can not be started; the goal manager might not be able to fulfill all requirements, it might be impossible to find or cast a fitting character, and the goals might not be accepted by any of the fitting characters. What happens then is dependent on the reason the plot thread is started; currently, the only reason to start a plot thread is to introduce conflict, in which case a different conflict will be tried.

Besides the requirements regarding the story world, the plot thread can also contain some notion of what can happen to the elements introduced by fulfilling those requirements; for example, it can describe the situation in which the conflict is resolved. In the treasure conflict, it might describe the conflict as being resolved when one of the characters has the treasure, and the other character cannot reach the first character to try and steal it again. The specifics of this are discussed in section 4.2.

Goal Manager The goal manager is a new part of the plot agent. Goals can be assigned to it, at the moment only by the plot thread manager, which it then tries to achieve, similarly to how characters do this. It is primarily used to fulfill requirements regarding the world that are given by a plot thread, though it can potentially be used for any goal the plot agent might have.

The goal manager keeps track of a number of goals, described as world facts that need to be true (or explicitly false) in order to consider the goal achieved. It uses the partial order planner already in use by the character agents [Kruizinga, 2007], which usually tries to make a plan consisting of the operators the character can use to achieve his goals, as discussed in section 3.2. However, since the plot agent is not an actual entity in the story world, it cannot perform any actions, or any other in-character operators for that matter. Therefore, only events (i.e. things that 'just happen', and are not really performed by a character) and framing operators (i.e. additions to the world of which it is pretended that they have always been there) are used.

It is also possible to pass on a requirement on the plan itself; for example, it can be required for the plan to contain at least one event. If no plan can be made containing an event (even though it is perfectly possible to make a plan without one), this is treated the same as when no plan can be formed at all. This is useful in enabling an inciting moment when introducing a conflict, as illustrated further below.

Once the plan is made, the goal manager will execute it, similarly as to how the character agent does this; every new round, the plan will be adjusted to the current world state and the goal manager requests (the rest of) the plot agent to execute the first operator in this plan. If this is possible, the operator will

be executed. In the case of framing operators, more of than one of those can be performed during one round; since a framing operator adds something to the world of which it is pretended to have always been there, nothing really happens in the story, and no time passes between operators. With the events, more than one can be performed during the same round if they do not depend on each other. Because events do take time, it can take several rounds to perform all of them if they are somehow dependent on each other.

When assigning a goal to the goal manager, it is also possible to indicate what should happen if the goal is achieved, or if that becomes impossible (i.e. when no plan can be made). In the case of a plot thread, the introduction of the characters and their goals takes place and the thread is added to the thread manager once the goal has been achieved. Upon failure, the goal is removed from the goal manager. Note that elements introduced in partially executed plans are not removed upon plan failure; this is rather complex, and it is assumed that plans are rarely aborted, because they consist largely of framing operators and events that can be executed in parallel. To disrupt such a plan, a precondition of one of its steps usually needs to be invalidated in exactly the same round as the one in which the execution of the plan starts (because otherwise the plan could not have been formed), but before the step itself has been executed, the timing of which seems rather hard to achieve coincidentally. Therefore, plans are rarely aborted, thus not cluttering the world too much, not making it worth the effort to remove elements introduced by an aborted plan. Besides, the newly introduced elements might lead to an unexpected, though interesting, story development.

Plot Thread Manager The plot thread manager is a new element of the plot agent that keeps track of all the plot threads that have been started. Because a plot thread not only contains requirements regarding the world, but also contains some notion of what should happen once these requirements have been fulfilled, it becomes necessary to keep track of the thread once it has been started. For example, if no track is kept of all the started conflicts, it becomes impossible to make sure they have all been resolved.

With regards to starting plot threads, and therefore conflicts, the thread manager can just be seen as a list to which started plot threads are added. The details of what is done with them afterwards are discussed in section 4.2.

The thread manager is also responsible for starting additional plot threads, as discussed in section 4.3.2.

Fitting it together Now that the elements used to introduce conflicts have been described, we can discuss how this fits together. This is done by means of an example conflict. The conflict consists of two pirates, LeChuck and Guybrush, that both want to have the same treasure.

The conflict is described in a plot thread. The plot thread requires the following of the story world:

- One character that is a pirate and a protagonist.
- One character that is a pirate and an antagonist.
- There should be a treasure chest in the world.

- The protagonist has a treasure map.
- The protagonist wants the treasure.
- The antagonist wants the treasure.

Additional information the plot thread might contain, like when the conflict is resolved, is ignored in this section, and will be treated in section 4.2.

The following world knowledge facts and operators (relevant to this example) are assumed to already exist:

- Guybrush is a character that is a greedy pirate and a protagonist.
- LeChuck is a character that is a pirate.
- There exists a framing operator that places a treasure in the world.
- There exists an event in which a character is given a treasure map.
- There exists a framing operator that makes a character greedy.

When the plot thread is started (for example, at the beginning of the simulation), the following happens:

- The existence of a treasure, and the protagonist having a map are passed as a goal to the goal manager. It is required of the goal manager to include at least one event in the plan to fulfill this goal. This makes sure something really 'happens' in the story (as opposed to just having framing operators), thus realizing some inciting moment.
- The goal manager tries to make a plan that adds the treasure to the world, and the map to the protagonist, using at least one event. If there already is a treasure, that treasure will be used. However, in this example that is assumed not to be the case. If the protagonist already has a treasure map, that one will be used. Again, this is assumed not to be the case.
- The goal manager makes a plan that consists of the use of the framing operator that places a treasure in the world, and the event that gives the map to the protagonistic character. If the character already had a map, the event would not have been used. This means that a plan would have been formed without an event, which was required, meaning that this conflict could not have been introduced.
- The first step of the plan is performed, and a treasure is introduced.
- The second step of the plan is performed, and an event occurs in which Guybrush gets the map. Because it is an event, as opposed to a framing operator, it indicates that something 'happens' in the story world, thus forming an inciting moment.
- Both requirements are fulfilled.
- The characters are being cast. There are two characters required, and both need to be pirates. One of them, Guybrush, was already used for the map finding event, and will be cast as one of the characters. In this example, it is assumed that no other pirate character has been cast before the plot thread was started, so a new character, LeChuck, is introduced, and a character agent is started to play his role.

- Both characters are offered the goal to find the treasure. The goal requires the character that takes it to be a greedy pirate, which happens to be the case with Guybrush. For LeChuck, a framing operator is performed that makes him greedy. They accept the goal, and start their treasure hunt.
- All elements needed to start a conflict have been introduced. The plot thread is added to the plot thread manager, to make sure the conflict is followed through.

Now we have a story world with Guybrush and LeChuck, two pirates that are looking for the same treasure, causing them to be in conflict with each other. Now that the conflict has been started, it can be tried to be resolved, as discussed in the next section.

Figure 4.2: Guybrush Threepwood, a mighty pirate.
Copyright 1997 LucasArts

4.2 Resolving Conflict

When a conflict has been introduced, as discussed in the previous section, it needs to be resolved in some way, in order to give the story some form of closure; the readers should have some idea how the story ends.

This section will first discuss when a conflict can be seen as resolved, after which a method is discussed concerning how this resolution can be brought about. This answers the second research question, 'How can the resolution of conflicts be facilitated?'

4.2.1 When is a conflict resolved?

In order to resolve a conflict, it should be defined when a conflict can be considered resolved. A naive approach would be to state that a conflict is resolved when one of the characters participating in it has achieved his goal corresponding to the conflict. However, this ignores the possibility that the other character can achieve his goal at a later time. For example, if Guybrush has achieved his goal of getting the treasure, LeChuck can still just kill him and steal the treasure, thus changing the outcome of the conflict.

A better method is to consider a conflict as resolved when one of the characters has achieved his goal, while the other character cannot find a plan to achieve his own; this way, it is clear that one of the characters has won the conflict. For example, if Guybrush has the treasure, and then leaves the island with the only ship available, thus leaving LeChuck stranded, it becomes obvious that Guybrush has won. However, this method has two drawbacks. The first one is that the other character might still be able to find some incredibly convoluted plan to achieve his goal, even though a normal audience will find the conflict to be resolved. In the previous example, LeChuck might build a new boat, train some (blood-thirsty) monkeys to be his crew, and try to follow Guybrush again; this plot seems better suited as a sequel to the treasure hunt adventure, and thus a new conflict, than being the same one. However, since LeChuck is still able to find some plan, the conflict will not be considered resolved.

The second drawback is that the planner cannot make plans that try to make the other character incapable of making a plan; therefore, characters are incapable of making plans to win a conflict. While it might be possible to adjust the planner, this is rather complex and time consuming, and has therefore not been attempted within this research.

Since this general approach does not work, a different method is devised that takes the specifics of the conflict into account. The method used in this research to detect the resolution of a conflict is to just state one or more conditions in which the conflict is resolved in the plot thread describing said conflict. In the treasure hunt case, one of the conditions might state that the conflict is resolved when the character with the treasure is on the main land, while the other character is stuck somewhere. The more abstract 'stuck somewhere'-part of the condition can be defined in the setting; a character can be considered 'stuck somewhere' if he is on an island, and there are no ships around, but if the setting includes a prison, a rule stating that a character is 'stuck somewhere' when he is in prison can be included. When one of the described conditions has been met, the conflict can be considered resolved. These conditions are called 'resolution-goals' in the rest of the text, since they are assigned to characters as

goals to meet; this will be discussed in more detail in the next section.

This method has two main advantages. First, the characters are capable of planning towards the resolution of the conflict, instead of just hoping that it resolves itself. Second, the author can specify under what conditions the conflict is resolved, solving the problem that the conflict keeps on running if the antagonist has a very improbable method of still winning the conflict.

Unfortunately, the latter advantage is also a disadvantage; because the author needs to state the specific conditions in which a conflict is resolved, its outcome becomes rather predictable. This is somewhat compensated for by the inference operator (discussed below), because it allows the author to state the outcome in more abstract terms (e.g., the antagonist just needs to be stuck, but it does not really matter where or how).

Inference Operator The inference operator is a new inw-character operator that can be used by the planner. It is used to draw conclusions about the current storyworld. It does not really change anything within that world, but it does change the interpretation of that world. In the example above, an inference operator can be written that states that when a character is on an island, and there are no boats in the neighborhood, then that character is stuck. When Guybrush is working towards a resolution-goal that states that LeChuck should be stuck, this inference operator can be used; Guybrush works towards the preconditions of this operator (i.e. LeChuck is on the island, and there are no boats around), and when the preconditions have been achieved, the inference operator is used, thus achieving the resolution-goal. As far as the reader is concerned, the inference operator has not changed anything about the physical world; storywise, LeChuck is not stuck on the island due to the inference operator, but due to the lack of boats. It has changed the interpretation of the world; LeChuck is now stuck on the island, instead of just being on an island without boats. This way, it can change the story itself; the precondition for building a raft might state that the character building it may not be stuck, as per inference operator, causing him to be incapable of leaving the island. However, the reader does not know that the character cannot build a raft due to the operator, and will (hopefully) assume that it could not have been built in the first place.

This operator has two main benefits; first, it makes it possible to capture a complex situation in a single fact. Instead of having to write that a character needs to be on the island without any boats, it is possible to just state that he needs to be stuck. This allows characters to plan towards a more abstract concept (i.e., being stuck) instead of the specific situation. Second, it allows for multiple situations in which the same conclusion can be drawn; besides just being on the island, a character can also be considered stuck when in prison. Instead of modifying every occurrence where a character needs to be stuck to include the prison situation, a new inference operator can just be added.

It should be noted that an inference operator needs to be explicitly used by a character (or the plot agent); the conclusion that someone is stuck will not be automatically drawn when someone is on an island without any boats. This might be seen as disadvantageous. Because the same conclusion is not always drawn in the same situation, it can lead to inconsistencies between stories, or even within the same one; LeChuck might be 'stuck' in a situation in which

Guybrush is not. However, this is mostly solved by making it implicitly known whether the conclusion an inference operator draws is important to the story; if no inference operator is used that concludes that a character is stuck, it does not matter to the story since no one is trying to get that character stuck. This can for example be used to allow or disallow certain actions; if, at the end of the story, LeChuck is thrown in prison (and this is part of Guybrush's resolution-goal), the author probably wants to keep him there (at least, until the sequel). If, on the other hand, in his adventures to get the treasure, Guybrush gets thrown in the prison, he might be allowed to use a framing operator that creates a dog that brings him the keys to said prison. By stating that a character should not be stuck as precondition to this framing operator, LeChuck will not be able to use this operator since Guybrush has used the inference operator to get him stuck in there. Guybrush, however, will be able to frame the dog, since at this point in the story nobody needs to be stuck, so the inference operator has not been used yet.

Inference operators are pretty similar to framing operators, in the sense that they both change something in the story, without something really happening. The main difference is that when performing a framing operator, it is assumed that its result has always been the case; i.e. the cannon did not just appear in the ship, it has always been there. The results of an inference operator, however, are not required to make any sense before the operator has been executed; it is not required for LeChuck to have been stuck the entire story in order to make him stuck by inference operator. Another difference is what happens with the results; the results of a framing operator can be changed by pretty much any kind of operator (e.g., the cannon can be moved), while the results of an inference operator are either involatile (i.e. LeChuck remains stuck), or can only be changed by another inference operator (since it concerns an abstract concept, and inference operators are the only operators that have effect on those concepts).

4.2.2 Working towards a resolution

When trying to solve a conflict, the characters involved will try to fulfill their goals. Obviously, it is possible that characters can resolve a conflict 'accidentally', meaning that the actions they take in fulfilling their goals also fulfill the resolution-goal, thus resolving the conflict. In this case, the Plot Agent does not need to intercede in any way.

However, if the conflict does not resolve itself, or it just takes too long, interference of the Plot Agent is required. To determine how to do this, the two main reasons (as identified by me) why a resolution does not occur automatically are examined.

First of all, the resolution-goal might differ from the character's main goal; for example, Guybrush' goal is just to get the treasure, while the resolution-goal also requires LeChuck to be stuck on the island. This problem is rather easy to solve though; as soon as Guybrush has the treasure, and has thus achieved his goal, he is suggested the resolution-goal as a new goal, which he will then try to achieve. In this case, he will try to leave the island, while trying to leave LeChuck stranded. If there are multiple resolution-goals, the one which requires the shortest plan will be chosen, as this is the one nearest to completion, and presumably the most logical one to be fulfilled. Obviously, when characters get

some emotional model, a different condition might be chosen; while it might be the shortest plan to kill LeChuck, Guybrush might have some qualms with this, preferring to turn LeChuck over to the authorities.

It is possible for characters not to accept their resolution goal. Because, at the time of this research, there is no model yet that characters use to either accept or refuse goals, it is not yet possible to cope with this. However, it can be assumed that the resolution goals are authored to be pretty logical follow-ups (for both the characters and the reader) to the main goal, otherwise the story will not make much sense. Therefore, if the main goal gets accepted, it makes sense that the resolution goal also gets accepted, or even autonomously adopted by the character. If a character wants to have the treasure, it makes sense that he wants to take it somewhere safe from other characters that also want the treasure.

Secondly, the characters might not have the means to achieve their goals. For example, if Guybrush has the same capabilities as LeChuck, it might be rather hard to escape from him. Also, Guybrush might be stuck on a different island (chased by blood-thirsty monkeys) while LeChuck can leisurely sail to the treasure and take it. To this end, the protagonist (assuming he is the winner; see below) is given additional operators when the conflict needs to be resolved. These climactic operators enable the protagonist to perform actions that might normally not be possible, or just with a very low success rate[1], or enable events and framing operators normally not possible. Examples are an action that kills the antagonist with a 100% success rate, a gust of wind-event that blows the protagonist's ship directly to the right island, or a framing operator that spawns an army of blood-thirsty monkeys that attack the antagonist. These are implemented as normal operators, except that they include a precondition checking what dramatic phase the story is in according to the plot agent. Since the conflicts should be resolved after the climax, these actions can only be used when the current dramatic phase is the falling action (discussed in section 4.4).

In order to give one character the advantage over the other, it needs to be determined beforehand which of the characters is allowed the use of the climactic operators. The text above assumes that this is the protagonist, but if a sad ending is wanted, the antagonist can be allowed the climactic operators instead.

Note that this system can also be used for operators during other phases, e.g., allowing some framing operators to only occur during the exposition phase, or making things a bit easier for the antagonist, and therefore tougher on the protagonist, during the rising action.

Because these actions are only allowed, not enforced, on characters, the plot agent helps in solving the conflict without taking away the character's autonomy.

Attention should be paid when authoring these climactic operators in that they should not model trivial actions (e.g., walking around, picking stuff up), because this can lower believability; readers might start wondering why a character did not perform that action earlier. This is less of a problem in the case of climactic events, because these just model lucky turns of events for the main character; no one is going to wonder why the antagonists ship did not suffer a crash earlier in the story. Climactic framing operators are a bit tricky; only

[1]It should be noted that there are no non-deterministic operators possible at the time of this writing. This is discussed in section 4.2.3.

being able to frame a cannon at the beach after the climax is probably okay, since it will be interpreted by the reader as it just being discovered by the protagonist, but only being able to frame a cannon on board of his ship after the climax will look really weird if it the cannon would have been usefull earlier in the story. Also, the cannon was considered to be always there, which can cause weird behavior if the antagonist does not pick it up while looking for a cannon (since it has not been framed there yet).

Both discussed methods, i.e. giving the resolution-goal as a goal and the climactic operators, place the responsibility of resolving the conflict on the protagonist. However, theoretically, the antagonist can also participate, by failing to perform some action, or choosing a plan that is not the best one, but does make things easier for the protagonist. This requires a separation of the character, that chooses to do what is best for himself, and the virtual 'actor' playing the character, who chooses actions based on what is best for the story instead. This method also raises questions like when characters can believably fail at actions, and how to determine whether a plan makes sense for a character, while not being the best plan. Due to time limitations, this method is not explored further.

4.2.3 Planning Issues

While the sections above discuss how it can be ensured that a conflict is somehow resolved, the story will still be rather boring if nothing interesting happens between the introduction of the conflict and its resolution. Currently, there are two major problems with the Virtual Storyteller that need to be addressed to make the stories more exciting. First, characters are barely aware of each other. Second, actions always produce the outcome a character expects them to produce. This chapter discusses these problems, and how they can be dealt with. However, due to time constraints, no real solutions are implemented.

Planning with multiple characters First of all, the characters have no real concept of other characters; while they know that the other characters exist, that they are somewhere, can carry stuff, etc., they do not do not take into account that the other characters can actually influence the world. This creates two problems; characters cannot make plans involving other characters in a different situation than the current one, and characters do not see other characters as possible threats to their own plans.

An example of the first problem, characters being incapable of making plans that involve characters in a different situation than the current one, follows: LeChuck is on the beach, Guybrush is in the jungle. Guybrush wants to set a trap in the jungle, and have LeChuck walk into it. However, Guybrush cannot make a plan because LeChuck is on the beach, not in the jungle.

This can be solved in part by means of expectation schemas, as introduced in section 3.2. These schemas model the expectations a character can have regarding pretty much anything in the world, including the behavior of other characters. It looks similar to any operator schema, in that it describes a set of preconditions, and some effects, and can be used by the planner to fulfill preconditions for some other operator. However, unlike operators, the expectations cannot be executed; the effects only describe what the character expects to happen. In this case, an expectation schema can model that if Guybrush has

the treasure, and LeChuck has the goal of getting that treasure, LeChuck will eventually be at the same location as Guybrush. This expectation schema can be used to fulfill the precondition in Guybrush' trap plan, namely that LeChuck needs to be in the jungle. He does not know that LeChuck will be there (because Guybrush does not take into account that LeChuck can actually move), but he assumes so, so he still builds his trap. Because, meanwhile, LeChuck still wants to have the treasure, he will eventually move to Guybrush' location in order to steal it, causing him to fulfill the precondition for springing the trap in reality (as opposed to just Guybrush' expectations), thus allowing Guybrush to spring this trap.

There are still some unresolved issues with the expectation schemas, like when a character should drop an expectation (if, for example, LeChuck has set a similar trap for Guybrush on the beach, assuming he has a similar expectation, he will wait for Guybrush over there, instead of walking into the jungle, thus never fulfilling Guybrush's expectation), and when to use an expectation in a plan instead of an action (for example, if LeChuck is sailing towards the island and Guybrush wants to shoot him, he can either wait for LeChuck to arrive by choosing an expectation, or sail towards LeChuck's boat and shoot him there). An unexplored (partial) solution to this problem might be to let characters try to fulfill the expectations of other characters. Obviously, this is not always in their best interest (walking into a trap is usually a bad idea), requiring the character agents to reason out-of-character how the characters they control can make the story interesting, which is not trivial to do.

The other problem is that characters do not see other characters as a threat to their plans; if LeChuck is on a boat sailing towards the treasure, Guybrush does not see this as a threat to his goal, because LeChuck being on a boat in no way hinders Guybrush picking up the treasure. Therefore, Guybrush will not do anything to oppose LeChuck, even though he is a threat to his goal in reality. This causes characters to not really compete with each other, resulting in less interesting stories. This can be partially resolved by authoring goals and resolution goals that involve the other characters, like above mentioned resolution goal that states that Guybrush should have left the island with the treasure, while LeChuck is still stuck there; this will cause a semblance of competition and awareness of the other character, but it will still not make Guybrush attempt to hinder LeChuck in order to be the first person to reach the treasure. To solve this, characters need to be aware of each other's plans (though not always the entire plan, otherwise characters cannot outsmart each other), which will probably require a major overhaul of the planner, and has not been done due to time constraints.

Deterministic actions A second issue with making the stories more exciting is that all actions are modeled as deterministic. This especially poses problems in swash-buckling pirate-adventures, in which sword-fights occur all the time; if a character knows beforehand whether he is going to win or lose a fight, he will probably choose entirely different actions than when he is still at risk. To model this, two changes are proposed to the action schemas; actions can have multiple outcomes, and the outcome might differ from what the character expected.

The first change modifies the effects of an action. Instead of listing just one set of world changes, multiple sets are listed, with a probability factor of each

of these sets occurring. Then, in the planner, the cost of selecting an action is somehow modified by the probability factor of the hoped for outcome; if a character thinks he has a 90% chance of winning in a swordfight, the action is rather likely to be used. If a character has only 10% chance of winning, he will probably not use that action unless he is really desperate. When the action is performed, the plot agent will then select which of the effect-sets actually occurs. The most obvious method is to select this randomly according to the probability, but in some situations the plot agent might wish to select a result depending on how this influences the story.

The second change adds a new set of effects to an action, the override-effects. When they are included in an action schema, they describe what *actually* happens when an action is performed, instead of what the character's planner thinks that will occur. This cannot only model entirely different outcomes (e.g., rubbing a lamp will cause a genie to appear instead of making the lamp shiny), but also different probabilities; an overly confident character might think that he has 90% chance of beating a pirate in a swordfight, but in reality his chances are a lot slimmer. While making a plan, the character agent ignores these override-effects. However, when the action is executed, the plot agent will then use the override-effects (if present) instead to determine the actual results of the action.

This latter change can also be used to keep the main characters alive until the climax; while other characters may think that starting a swordfight with LeChuck can cause LeChuck to die, the override-effects make sure this cannot happen until the end of the story.

As has been stated before, these changes are just proposals, and have not been implemented or tested in any way.

Figure 4.3: LeChuck, an evil pirate and the villain of the stories.
Copyright 1997 LucasArts

4.3 Rising action

This section discusses the rising action, the phase of a story in which secondary conflicts are introduced. First, it is discussed what secondary conflicts are, followed by how and when they can be introduced in the story, thus answering the third and fourth research questions ('What makes a conflict a secondary conflict?' and 'When should conflicts be introduced?'). Finally, two remaining issues are discussed, namely the secondary conflict types that cannot (yet) be implemented with the proposed method, and what the antagonist should do during the protagonist's adventures.

4.3.1 Secondary conflicts

During the rising action phase of a story, secondary conflicts are introduced in order to increase the suspense and excitement of the story. These secondary conflicts are conflicts that form additional obstacles to the goal in the protagonist's primary conflict. They may be directly related to the primary conflict (e.g., the protagonist needs to find all four pieces of the treasure map in order to get the treasure, where the missing of each piece is a secondary conflict), or totally unrelated (e.g., the protagonist gets chased by a bunch of blood-thirsty monkeys).

Whether a conflict is a primary or secondary conflict depends on the context it is used in. The treasure hunting conflict that has been used as an example of a primary conflict can just as well be a secondary conflict, if for example the treasure is used as a means to fund achieving the protagonist's primary goal. Being chased by blood-thirsty monkeys is an obvious secondary conflict in a treasure hunt adventure, but can be used as a primary conflict in a horror movie.

While, due to personal preferences, the focus with primary conflicts has been on conflicts between two characters, a secondary conflict can be any obstacle in achieving one's primary goal, and therefore does not need to be specifically between two characters. For example, suffering a shipwreck is an obstacle in trying to find a treasure, even though no other character is directly involved. However, there is nothing in this research that prevents an author from writing a primary conflict without an antagonist, and secondary conflicts that include one.

4.3.2 Introducing the secondary conflicts

Obviously, secondary conflicts should be described in some way, in order to introduce them into the story. Since, as discussed above, there is no real difference between a primary and secondary conflict, plot threads can be used again. A marker can be added to the plot thread scheme to indicate that a thread can only be used as a primary or secondary conflict, in order to give the author more control over the story that is generated; if the author wants to create a story about a treasure hunt, it would be rather annoying if the blood-thirsty monkeys are chosen as the primary conflict.

Introducing the secondary conflicts into the story is conceptually different from the primary conflict though; the plot thread describing the primary conflict is started when the story starts (the starting of this thread can be seen as

the inciting moment, see section 4.1), while secondary conflicts are introduced during the story. This means that starting the primary conflict can shape the world in pretty much any way necessary to fit it, while the secondary conflicts can only change it in a way that does not interfere with what has already been established in the story. Fortunately, this is already handled by the way in which plot threads are started: the requirements of a plot thread are filled in by means of events and framing operators. Since it is not allowed to execute such an operator in a way that is inconsistent with the world, it becomes impossible to make a plan for the requirements of a plot thread that cannot be started in the current story.

This same mechanic can be used in choosing which plot thread to start when introducing a secondary conflict; the length of the plan created by the plot agent's goal manager to fulfill the thread's requirements is assumed to be a good indication of how well that thread fits in the current story. A smaller plan means that fewer operators are needed, and therefore fewer changes to the world, thus indicating that the plot thread meshes fairly well with the world.

Timing A more complicated issue than *how* to introduce the secondary conflicts is *when* to introduce them. Ideally, they should be introduced when the story starts getting boring (actually, just before then), but figuring out how to detect this is too complex and time-consuming to do this within this study. Therefore, a couple of simple guidelines can be used to select when a new secondary conflict should be introduced. Four have been implemented, but these are by no means conclusive:

1. As soon as the story (i.e. the primary conflict thread) is started; this gives the protagonist some immediately identifiable obstacle in achieving his primary goal.

2. When the protagonist has not taken any action for a number of rounds, for example when waiting for an expectation to be fulfilled (see section 4.2.3); since a story about a pirate that just sits around, waiting for things to happen around him, is not that interesting, this will make sure something happens to him.

3. When the protagonist has almost finished his currently active goal (as measured in number of steps left in his plan). This makes it harder for the protagonist to achieve his goal, so it will cause more suspense.

4. If, at any time, no secondary conflict has been running for a number of rounds.

The specific guidelines to be used can be configured by the author, according to taste, or experience regarding which of the guidelines work in a specific type of story. When one of the configured guidelines is met, the plot thread manager will try to start a new secondary conflict.

Also, secondary conflicts should only be introduced during the rising action, so as soon as the climax has started (see section 4.4), no new conflicts should be started.

It might be impossible to start a new conflict exactly when one of the guidelines above applies (if the requirements of none of the authored secondary con-

flict plot threads can be fulfilled); if that is the case, the conflict is started as soon as possible after the guideline has been met.

Examples An example of how this fits together follows.

Guybrush is on a quest to find a treasure (his primary conflict). Two secondary conflicts exist in this story domain; the first conflict describes that the protagonist is on an island that is not the island with the treasure, with no immediate way to leave, and has a goal that states that the protagonist should be off this island. The second conflict describes a bunch of blood-thirsty monkeys on any island that is currently occupied by the protagonist. When the story starts, Guybrush is in a harbor somewhere on the main land (i.e. not an island). Since the story has just been started, a secondary conflict should be introduced as soon as possible, following guideline 1. Guybrush arranges a ship, and sets sail to Monkey Island, the island that contains the treasure. As soon as his ship is sailing, the plot agent's goal manager finds a way to fulfill the conditions of the first conflict; an event can be executed that blows the ship to any island, and destroys it in the process. This event is executed, and Guybrush strands on Dinky Island. Here, he starts repairing his ship in order to get off the island. When he is almost finished, a new secondary conflict is started, following guideline 3; the blood-thirsty monkeys are introduced, and start chasing Guybrush around the island.

A different example shows how these secondary conflicts can also directly influence the primary conflict.

Guybrush starts on a quest to find a treasure. To find this treasure, a map is needed, which has been added to the storyworld by the plot thread of the primary conflict. As soon as the story is started, a secondary conflict is introduced, which has the following requirements:

- There is no treasure map[2].
- The protagonist has the goal to find a treasure.
- There are four map parts divided between three islands and the protagonist.

When this secondary conflict is started, a framing operator is used that changes a map into a map-part[3], indicating that the map was actually a map part, as well as three operators that add the other parts to various islands. This way, the primary conflict (finding the treasure) is complicated due to the map actually being split into four pieces.

Note that these latter secondary conflicts are more complicated to write, and also require some very convoluted framing operators. This suggests that this approach may not be ideal for conflicts that directly influence the primary conflict, and some might even be impossible to script (for example, Guybrush being lured to a decoy treasure instead of the real deal, discussed in the next section). However, even though there are limits to the conflicts that can be

[2]This might seem counter-intuitive, but the requirements in a plot thread state the condition the requirements on the story world *after* it has been started, not before. See section 4.1.

[3]This framing operator has as an additional requirement that no one has taken a look at the map, because it would otherwise mean that the map has been established as the entire map within the story.

written this way, it should still be possible to come up with lots of different conflicts with this method.

The conflicts that cannot be written, and what can be done about it, are discussed in more detail in the next section.

4.3.3 Other issues

This section discusses two remaining issues with the previously presented approach to secondary conflicts. First of all, it discusses what the antagonist should be doing during the protagonist's hijinks. Second, the limits of the possible secondary conflicts are discussed in more detail.

The Antagonist's Adventures When the story's protagonist gets stuck in all sorts of secondary conflicts, there should be some reason why the antagonist does not 'win' the primary conflict in the meantime. In the treasure hunt example, why would LeChuck not just keep on sailing to Monkey Island to pick up the treasure while Guybrush is stuck with blood-thirsty monkeys on Dinky Island? Why would LeChuck not just marry the kidnapped pirate princess, while Guybrush is trying to rescue her?

The latter example is pretty easy to explain within the story, without too much difficulty; LeChuck is still working on e.g., the wedding preparations, giving Guybrush enough time to deal with his crazy adventures. Within the storyteller, this can be solved by making the 'force-princess-to-marry'-action a very long action that can only take place after the climax; this way, the wedding starts after Guybrush has dealt with most of his problems (see section 4.4), while still giving the reader the feeling that he is on a race against the clock to stop the wedding. This is rather similar to how TAILOR [Smith and Witten, 1987] makes characters skip turns in order to ensure that they interact.

This approach does not work in the first example; the equivalent would be to allow LeChuck to arrive on Monkey Island only when Guybrush has arrived, which would imply that LeChuck is a very slow sailor, which would not be believable (though it can be argued that the reader does not really care about the antagonist's hijinks, and that the LeChuck's sailing speed just represents the adventures he gets into). An obvious approach would be to just get the antagonist involved in some secondary conflicts himself. This has the risks that the antagonist is still stuck in *his* secondary adventures, while the protagonist is winning the primary conflict without too much resistance. This can be solved by allowing climactic actions for the antagonist, but only when not used to achieve his primary goal. However, this would reduce believability because, to the reader, there is no reason why the antagonist does not use his climactic superpowers when dealing with the primary conflict. Also, the antagonist's secondary adventures make the protagonist's reversal of fortune in the climax less dramatic; the protagonist is not faring that much worse than the antagonist, so it feels less like he has to go through insurmountable odds to achieve his goal.

Because both of these approaches have advantages and disadvantages, depending on the situation, no generic solution is given. However, they can both be used at the moment, according to the author's wishes.

Unscriptable Conflicts The second issue concerns the secondary conflicts that are not possible to script at the moment. The problems with the two types

of conflicts discussed here are the ones that were encountered while trying to write secondary conflicts over the course of this research, though there might be more. The first is that of a 'false knowledge', i.e. the primary goal that the protagonist was working to turns out to not be his actual goal. The second is that of adding additional requirements to achieving the primary goal.

False knowledge The first secondary conflict, the 'false knowledge' is easily described in an example. When Guybrush has finally gotten the treasure, it appears that the chest does not contain the treasure at all, but, for example, a map to find the real treasure. Similarly, when Guybrush breaks into the castle that should contain the pirate princess, it turns out that the princess is in another castle.

These conflicts cannot be scripted due to the characters knowing everything about the world; if the contents of the treasure chest are changed when starting a secondary conflict, the protagonist will know that it now contains a map instead of a treasure. When moving the princess to a different castle, the protagonist will not bother checking the fake castle. Therefore, this problem can be solved by allowing the characters to have beliefs that are different from the reality. Then, these kind of secondary conflicts can be scripted by either changing the world without updating the character's belief (e.g., change the contents of the treasure chest), or to change the belief a character has (assuming he has not acted upon it), like changing his belief concerning the castle the princess is at.

At the moment, the character agents are already implemented that way; they have their own beliefs and perceptions that do not necessarily correspond with reality (i.e. what the world agent says). Beliefs and reality currently match because every change in the world is directly passed through to the characters; by hiding the execution of the framing operator that (for example) replaces the treasure with the treasure map, the character would not know it has changed, and will still try to reach the original treasure.

While this might seem trivial to implement, it raises some issues that are more complex; regarding the characters learning the truth. For example, when will the character learn that he has just a treasure map instead of the treasure? Only when he opens the chest (causing him to see the map), or will he notice that something is wrong when lifting the treasure chest (that weights far too little to contain massive amounts of coins)? Also, different perceptions might conflict; when Guybrush accidentally finds the princess in her new location, instead of the location he expects her to be, the belief that the princess is in the expected place needs to be erased. However, if he finds the treasure on a different location than expected, he might just as well assume that it is a different treasure. Some method of discerning what a character notices and what not needs to be implemented as well; the true location of the princess might be revealed if LeChuck interacts with her, even though Guybrush is on an entirely different island.

Due to these issues, there is not enough time to solve this within this research.

Adding requirements The second type of secondary conflict, adding requirements, is rather difficult to implement, because it involves changing schema's of, for example, actions during the simulation. For example, when looking for treasure, it could be that a secondary conflict adds that a map is needed to

find the treasure. This would probably involve setting an additional precondition to the action that sails the ship to Monkey Island (where the treasure is), namely that of having the map[4]. When Elaine, the pirate princess, has been turned into a gold statue by a cursed ring, Guybrush needs to recover that ring to reverse the curse. However, in a secondary conflict, it suddenly turns out that not only the ring is needed (e.g., because the ring alone does not appear to work, or a voodoo priest tells this to the character), but also the ingredients of a mysterious spell. The seemingly easiest solution would involve adding the spell ingredients to the preconditions of the 'reverse the curse'-action.

The conceptual solution to this is rather trivial; just add an additional precondition to an action. However, this is less than trivial to implement; action schema's are fixed at the moment, and are not really meant to be modified within a running simulation. While it might be possible to change this, it would probably require a complete overhaul of the current schema system.

Figure 4.4: Elaine, the lovely pirate princess.
Copyright 1997 LucasArts

[4]It should be noted that this particular secondary conflict can also be implemented by removing the characters knowledge where the treasure is (assuming he has not acted upon it), only allowing him to regain it when having the map.

4.4 Ending the story

With the exception of some soap operas, every story comes to an end. In order to make the Virtual Storyteller generate a finished story, some method needs to be found to give an end to it.

According to Freytag's pyramid (section 2.4.1), the story 'starts ending' with the climax, when things finally end up going well for the protagonist. This is followed by the falling action, in which all unresolved conflicts (both the primary and all the secondary ones) get resolved, after which the protagonist gets to enjoy the outcome of the conflicts in the dénouement phase. What these phases mean within the Virtual Storyteller, as well as how to implement them, is discussed in the sections below, giving answers to the last two research questions, 'When is the right time to resolve the conflicts?' and 'When does a story end?'.

4.4.1 Climax

As has been mentioned in sections 4.2.2 and 4.3.2 respectively, the climactic phase in the Virtual Storyteller basically boils down to two things:

1. Enabling climactic actions.
2. Halting the introduction of new secondary conflicts.

The climactic actions give the protagonist additional options in achieving his goals, making it possible (or just easier) to solve the conflicts. Halting the introduction of new conflicts makes sure that no new conflicts appear to make the protagonist's life difficult, causing a true reversal of fortune. Also, it becomes rather impossible to resolve all conflicts if new ones keep on appearing.

While we have already discussed *what* should happen during the climax, it has not been determined yet *when* the climax should occur.

For this, three heuristics are used; whenever one of them is applicable, the climax is started.

The first heuristic is met when the protagonist cannot find a way to solve any of his goals; this is the preferred method, since it indicates that the protagonist is truly in trouble he cannot find an easy way out of, thus causing a real reversal of fortune when he uses his climactic actions to get out of the trouble.

The second heuristic is met when any of the goals of the primary conflict are achieved; this allows for a climactic, exciting showdown between the characters when they are trying to achieve their resolution goals.

The final heuristic is useful for when the above situations never occur, or take too long to occur; after a certain (as determined by the author) number of secondary conflicts have been introduced, the story has been going on for long enough, and should be brought to an end by starting the climax. This way, the author has some control over the length of the generated story. It also assures that the story does not get 'stuck' (i.e. nothing really happens), while the characters are still capable of making plans (thus not triggering the first heuristic). This latter situation can for example occur when a character is waiting for an expectation to be resolved that will never be resolved (discussed in section 4.2.3), or when two characters keep on switching locations with each

other when planning to go to each other, while never meeting. It is still possible for such a situation to occur, but this way, it does not keep on going indefinitely.

Note that the climax is not really a phase the story is in, but just the tip of the (Freytag's) pyramid; the climax enables the climactic actions, and stops new secondary conflicts from appearing, but then the story immediately moves on to the falling action.

4.4.2 Falling action

After the climax has occurred, the falling action takes place, in which all open conflicts get resolved. As described in section 4.2, the plot and character agents already try to resolve the running conflicts, and the climax makes sure that the characters actually have the means to resolve them. This means that the falling action pretty much takes place automatically, without requiring any extra work.

4.4.3 Dénouement

When all conflicts have been resolved, the protagonist gets to enjoy his reversed fortune in the dénouement. Unfortunately, due to the lack of emotional model, the characters cannot really enjoy anything yet, so, at the moment, the story just ends when all conflicts have been resolved.

However, the dénouement becomes rather easy to model when the characters actually can enjoy things. A set of actions can be used that tie in to the main conflict (or pretty much any conflict where it makes sense). These actions can only be used during the dénouement, and give the character that performs the action great joy (or any other emotion that makes the character want to use the action). For example, an 'enjoy the treasure'-action can be written, that requires a the protagonist to have a treasure in his possession. Due to the emotion affected by this action, the character will use it as soon as possible. Since this is only allowed during the dénouement, it will only be used at the end of the story, causing it to end on a happy note. A different 'marry the pirate princess (but not against her will)'-action can be used in the adventure where the pirate princess was rescued. If a secondary conflict in the princess-adventure requires the treasure, both of the above actions can be performed for even greater enjoyment.

While it would have been possible to just give characters the goal to perform a specific dénouement-action, even without the emotional model, this would have been somewhat of a hack, and has therefore not been implemented.

Figure 4.5: Elaine and Guybrush, happily ever after.
Copyright 1997 LucasArts

4.5 Summary

This section summarizes the previous sections by giving an overview of what should happen during every story phase, thus indicating how Freytag's pyramid is followed.

Since Freytag's pyramid indicates what happens to conflict within a story, and the conflict is represented with a plot thread in the Virtual Storyteller, a summary of what a plot thread contains is given first:

- **Name** Plot threads have a name, so that they can be referenced more easily.
- **Type** The type of the plot thread. This thesis differentiates between three types: *primary conflict*, *secondary conflict*, and just *conflict*, when it does not matter if the described conflict is started as a primary or secondary conflict. The number of types can theoretically be expanded for additional uses, besides conflicts, of the plot thread.
- **Requirements** The plot thread contains a list of requirements that the world should meet in order for the plot thread to start. These requirements can contain facts about the world, needed characters, and goals those characters should have.
- **Resolution-goals** The plot thread can optionally contain resolution-goals that characters should take on when they have achieved their primary goal (the one stated in the requirements) of this plot thread.

The following table describes what happens during every story phase. The first column indicates the story phase (or point) in Freytag's pyramid, while the second gives the name of the story phase within the Virtual Storyteller. The last column indicates what happens during that phase; this is described in more detail following the table.

Freytag	VST	What happens?	Duration
Exposition	`exposition`	The story world is started, and the requirements of the plot thread of the primary conflict are being fullfilled.	Phase
Inciting Event	`rising_action`	The requirements of the primary conflict have just been fullfilled; character agents start making plans.	Point
Rising Action		The character agents try to follow their plans, while secondary conflicts are being introduced.	Phase
Climax	`falling_action`	From this moment on, no more new secondary conflicts are introduced, and climactic actions are allowed. Usually occurs when one of the characters has achieved the goal of his primary conflict.	Point
Falling Action		All still running conflicts are being resolved.	Phase
Dénouement	`denouement`	All conflicts have been resolved. The story ends.	Point[5]

Exposition During the exposition, the following happens:

1. The story world settings are loaded.
2. The story phase is set to `exposition`.
3. A plot thread containing a primary conflict is selected by the plot thread manager. The selection criterion is that a plan can be made to fulfill the thread's requirements; if more threads can be found, one is chosen at random.
4. The plot thread manager passes the thread's requirements on to the plot agent's goal manager.
5. The goal manager fulfills these requirements by means of framing operators, events and inference operators, following the plan made in step 3.

The exposition is just a set up for the rest of the story, and therefore very little fabula should be generated. The characters themselves are not active during this phase (because no agents are assigned to the characters), so the only fabula that can be generated comes from events that are used to fulfill the plot thread's requirements, indicating that something happens to the characters that causes them to go on their adventures (i.e. the inciting event).

Inciting Event The inciting event is a point after which the adventure starts. It can be an event (operator) used to fulfill the primary conflict's requirements (in which case fabula is generated concerning this event, making it possible for the reader can see what happened to have the adventure start *now*); if no such

event is used, the inciting event is not generated in the fabula, which means the reader will have to fill it in on his own. This probably is not that much of a disaster in the stories that are currently generated by the Virtual Storyteller.

During the inciting event, as seen by the Virtual Storyteller, nothing really happens; it just marks the end of the exposition, and does not require any time (within the story). The story phase is set to `rising_action`, the character agents are associated with characters, their goals are assigned and the characters will start making their plans for the rising action.

Rising Action During the rising action, the following things happen simultaneously:

- The character agents try to fulfill their goals; at first only the goals stated in the primary conflict, but goals from secondary conflicts might be added when such a conflict is started.
- Secondary conflicts are started; at certain moments, as configured by the author (see section 4.3.2), the plot thread manager tries to start a secondary conflict. The criterion is that the requirements of the plot thread describing that conflict can be fullfilled. If multiple threads are possible, the one with the shortest plan is selected. The requirements of that thread are passed to the plot agent's goal manager.
- The goal manager tries to fulfill the requirements of any pending secondary conflict. If those requirements are met, the goals associated with that secondary conflict are assigned, and any additional characters are started.
- If, during the rising action, a goal has been achieved by a character, and the conflict that goal originated from describes a resolution goal for that character, the resolution goal is assigned. If that goal originated from the primary conflict, the climax is started.
- If the protagonist is incapable of fullfilling any of his goals, the climax is started. (If this situation occurs earlier than a main goal of the primary conflict being solved.)

The during this phase, the story phase remains `rising_action`, as set by the inciting event.

Climax The climax, like the inciting event, is just a point, not a phase; as far as any observer is concerned, the story goes directly from the rising action to the falling action. The climax does the following things:

- The introduction of any additional secondary plot threads is stopped.
- The story phase is set to `falling_action`. This has the additional effect of allowing climactic operators (as described in section 4.2.2), since these require the story phase to be the falling action to execute.

Hereafter, the falling action starts.

Falling action During the falling action, the characters try to achieve their still remaining goals, and the resolution goals associated with any conflict they got dragged into. This keeps on going until all conflicts are resolved, meaning:

- If the corresponding plot thread has resolution goals associated, if any of these resolution goals is achieved, the conflict is resolved.
- If the plot thread does not have any resolution goals, achieving any goal associated with the thread resolves the conflict.

Note that, obviously, during this phase, the story phase is still set to `falling_action`, allowing climactic actions.

Dénouement In the dénouement, the story phase is set to `denouement`. Besides this, nothing happens, and the story is finished.

Figure 4.6: A blood-thirsty monkey. Yes.
Copyright 2004 Rik Shepherd and Mr Monkey.

Chapter 5

Authoring

With chapter 4 having introduced all the required elements to create a story with a dramatic structure in the Virtual Storyteller, this section will now discuss how these elements can be used to actually author a story. First, some useful operators that can be used with any story are given. Then, the Treasure Hunt Story, used as an example throughout the previous section, will be used to illustrate how this can be done, after which the Rescue the Pirate-Princess Story will be written, to show how different stories can be written with the same techniques.

Note that this section is not a guide to authoring stories with the Virtual Storyteller, and therefore focuses on the new elements introduced in this thesis.

Actual stories that can get generated with the authored story contents written in these sections are shown in chapter 6.

5.1 Operators

As described in section 4.1, plot threads are introduced by fulfilling the requirements associated with them. These requirements are fulfilled by means of operators. A couple of operators that are useful in any pirate story are given in this section. Coincidentally, these operators all happen to be framing operators.

SelectProtagonist The `SelectProtagonist` framing operator makes a character a `protagonist`, if no `protagonist` already exists, and the character is not an `antagonist`.

SelectAntagonist The `SelectAntagonist` framing operator makes a character a `antagonist`, if no `antagonist` already exists, and the character is not a `protagonist`.

PlaceCharacter The `PlaceCharacter` framing operator places a character anywhere in the world, assuming the character does not have a location yet.

MakePirate A character becomes a `pirate`. Any reasons why a character cannot become a pirate should also be noted here (e.g. he is already a `lawyer`), but no such exceptions exist in this story world.

5.2 Treasure Hunt

This section discusses the authoring of a story about two pirates that are looking for treasure.

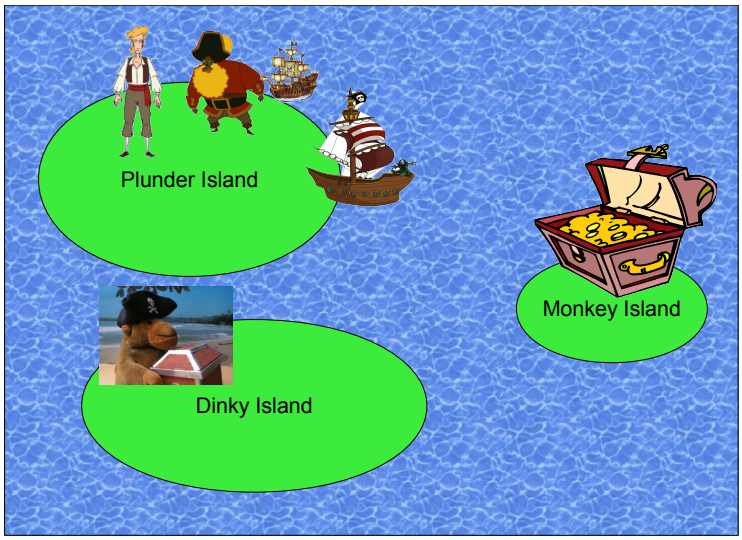

Figure 5.1: The map as it looks in the Treasure Hunt Story.

Starting the Main Conflict First of all, the main conflict of the story needs to be set up. In this case, the conflict describes a treasure hunt between two characters regarding a treasure that is on an island. Some parts of the world setting are added manually (as opposed to introducing them by plot thread):

- Monkey Island and Plunder Island, in order to have the treasure and the pirates' home base on different islands.
- There are two characters, Guybrush and LeChuck, but they are not assigned any location.
- There are two ships moored at Plunder Island, to allow the characters to move between the islands.

Note that any of these settings can also be introduced by stating them in the plot thread, and writing the required framing operators to introduce them. However, locations (the islands and, technically, the ships) and characters require a lot of separate facts to represent them, making the requirements of the plot thread and, especially, the corresponding framing operators, rather complicated.

Also, some basic actions are needed, like moving around, picking stuff up, and sailing the ships. Now we have a basic world, in which the wacky treasure adventure can take place.

Now, to get the adventure started, a plot thread, as described in section 4.1 needs to be written, describing all requirements for the treasure hunt story.

This plot thread will be called `TreasureHuntAdventure`. In this case, the requirements-list will look like the following:

- There exists a treasure on Monkey Island. (Obviously, otherwise there would not be a treasure to hunt.)
- There are at least two pirates in the world.
- The pirates are at Plunder Island[1]
- The pirates have the `GetTreasure` goal, with the treasure from the first requirement filled in as the specific treasure.

The `GetTreasure` goal is a somewhat simple goal that has the success-condition that the character having the goal holds a treasure. The specific treasure can be given as an argument of the goal, but otherwise a random treasure in the world will be chosen. A precondition to this goal is that a treasure exists.

The treasure requirement needs to be filled in somehow. This can be done by manually adding a treasure to the world (on Monkey Island), or by writing a framing operator that adds a treasure to the world. If the framing operator method is used, care should be taken that the treasure can only be introduced once, because otherwise the characters can just use it again to create a second treasure to solve the conflict. Besides this, it does not really matter which method is used, though the framing operator method is recommended because it is reusable, and can thus be used in other stories than just this treasure hunt adventure.

When the plot thread is started, the requirements will be filled in, creating the treasure if necessary, and giving LeChuck and Guybrush the goal to get the treasure. The pirates will now set sail to get the treasure.

Resolving the Main Conflict The plot thread described in the previous paragraph results in the pirates trying to get the treasure, which ends as soon as the first one to reach it has picked it up (because one of its goals is fulfilled). This isn't really that exciting, and does not feel like a finished story. In order to make things somewhat more interesting, the `Steal`-action is added, which allows a character to steal an object held by another character, assuming they are in the same location.

Unfortunately, this will result in a story in which the pirates end up just stealing the treasure from each other, without it ending. Therefore, a resolution goal needs to be written, as described in section 4.2.1, stating the conditions that need to be true in order for a conflict to be resolved. In this case, the goal is called `EscapeWithTreasure`. It has the precondition that the one having the goal has a treasure (meaning that the goal cannot be assumed without it, as it would not make much sense), and the following success conditions:

- The character following the goal has the treasure.

[1] The requirements to the locations can also be written more generally, as that none of the pirates is at the same island as the treasure. Directly indicating the location is somewhat simpler though, and increases the readability of the text (especially since more locations are going to be introduced in this section), so this convention is used.

- The character is on Plunder Island (his home).
- His opponent is `stuck`. (Discussed further below.)

The specific treasure and opponent are arguments that can be filled in by the plot thread. By indicating this goal as the resolved goal for the plot thread (with the treasure, and the characters' opponents as arguments), the pirates will try to escape with the treasure when they have achieved it.

The `stuck`-condition is one that needs to be filled in with an inference operator. It indicates that it does not really matter to the resolve goal how the opponent gets stuck, just that he somehow does. In this case, two examples are given, but more can easily be added.

The first inference operator is `MakeStuckOnIsland`. It has the following preconditions:

- The target is a character.
- The target is on an island.
- There are no ships moored at the island.

Its result is that the target gets `stuck`. It assumes that a character that is on an island, without ships, has no way of leaving the island, so the author should pay attention that this is actually the case.

Now a character with the `EscapeWithTreasure` goal will try to return to Plunder Island with the treasure, leaving his opponent on the island while there are no ships around. Obviously, some method is required to make sure that there are no ships on the island, which true pirates do with cannons.

To this end, a framing operator is introduced that creates a cannon on a ship (or they can be manually authored to be on the ship), as well as two actions, one of which loads the cannon, while the other fires a loaded cannon at a ship; when the ship is hit, it is both `destroyed` and not moored on the island anymore. The latter is more for convenience, as an inference operator can also be used that states that if a ship is `destroyed`, it is not moored at its current location anymore. In this case, it is important that the ship somehow isn't moored at the island anymore when it is hit by a cannon, because that is the condition that the `MakeStuckOnIsland` operator checks for. The location the ship is moored at (if any) when destroyed is also stored in a `destroyedAt`-fact; this is needed for the `TheDamageWasntThatBig` operator, described below.

A second inference operator is also needed, `MakeStuckOnShip`, that makes a character `stuck` on a `destroyed` ship, in case a pirate was already on board the ship when it was shot.

This should result in a story in which the pirates try to get the treasure, and the one with the treasure leaves the island in his ship, while blowing the ship of the other pirate up. However, some cases are not covered, that can result in an entirely different story. For example, both pirates can fire their cannons at each other's ship which will leave both pirates stranded. While this brings some moral into the story regarding the futility of violence, this probably isn't the desired outcome.

Fortunately, this can be solved with the climactic actions, as described in section 4.2.2. In this case, a climactic framing operator, `CreateLoadedCannon`, is used, that has the following preconditions:

- The character selecting it is the protagonist.
- The story phase is in the `falling_action`, meaning that the climax has passed, and that climactic actions can be used.

The result of this framing operator is that a cannon appears (anywhere; it can e.g. be found on the beach) that is already loaded, thus giving the protagonist an advantage in blowing up the other's boat (he can shoot faster). With climactic framing operators, care should be taken that it is still believable that the antagonist has not acted upon the result of the operator yet. Since the result of the framing operator is assumed to have always been there, nothing really happens when it is executed; the loaded cannon does not just suddenly appear. In this case, the antagonist not interacting with it before the operator is executed can be interpreted as him just not seeing it, or not needing it before. This does get more problematic if characters are capable of framing entire cities this way. In this case, this problem could have been avoided by making a climactic event or action in which the protagonist explicitly *finds* the cannon.

Note that the protagonist is selected with the `SelectProtagonist` operator. If none of the pirates explicitly needed to be a protagonist before one needs `CreateLoadedCannon`, the first pirate to use it becomes protagonist.

It is theoretically still possible for both pirates to destroy each other's ship at the same time, if the protagonist's decision to shoot comes later, thus compensating for his ability to shoot faster. This can still be solved in lots of ways, but for simplicity, we just add a `TheDamageWasntThatBig` climactic framing operator, that repairs a ship of the protagonist's choice by making it not `destroyed` anymore, and moores it at the location it was before it was destroyed (if existing). Note that these kind of operators (i.e. those reversing, or changing, something in the world, instead of just adding it) can be kind of tricky, because they require the characters to not have acted upon the previous situation in order to maintain believability. Though hacky, it will probably not be much of a problem in this story, because characters acting upon the destroyed ship will probably be interpreted as not noticing that the damage was not that big, i.e. that their perception of the ship changes instead of the reality surrounding it.

With this in place, a story can be generated in which both pirates will try to get the same treasure. After the antagonist has reached the treasure, he will try to race back to Plunder Island, while blowing up the other's ship. However, the protagonist was lucky enough to find a working cannon, and shoots the ship of the antagonist while he is trying to leave. He steals the antagonist's treasure, and leaves on his own ship, thus ending the story.

The action in the end can of course be spiced up by adding sword fights, allowing ships to be entered, etc.

Secondary Conflicts In the above story, a lot of stuff can happen once the treasure has been reached, but reaching the treasure itself is pretty straightforward. In order to do something about it, secondary conflicts can be added, as described in section 4.3.2. Authoring two secondary conflicts given in that section, i.e. stranding on an island, and the island being filled with bloodthirsty monkeys, is described here.

First of all, some modification is needed regarding the main plot thread in the previous paragraphs; it needs to be marked as the primary conflict. This

is to prevent it from being used as a secondary conflict, which is not what is wanted in this case. Threads marked as the primary conflict can only be started in the exposition, threads marked as secondary conflicts only during the rising action, and unmarked threads can be started during either.

Then we can add secondary conflicts. The first added is the crashing of the protagonist on an island. First, a third island (that shall be called Dinky Island) is added the same way as Monkey and Plunder island were. Then, we need some way of crashing a ship. To this end, the `CrashShipWithCrew` event is created. It has the following preconditions:

- A ship is somewhere at sea.
- There is a character on the ship.
- The location the ship is crashed upon is the shore of an island.
- That location isn't part of Monkey Island. (This is optional and is to prevent characters from using this event instead of the `Sail`-actions to reach the island, which is probably preferred.)
- The story is in the rising action. (This is to prevent characters from using it to destroy each other's ship during the resolution of the main conflict. If the author actually likes this, the precondition can be removed.)

As a result of this event, the ship is now `destroyed`, but moored at the chosen location. The character on the ship is now located at the location where the ship crashed (i.e. he was thrown off the ship)[2].

Then, a plot thread, `ShipWreckAdventure`, marked as describing a secondary conflict, can be written using the following requirements:

- The character is on Dinky Island.
- The character has the goal to be on a ship that is at sea.
- That ship is currently destroyed, but still moored at Dinky Island.
- The character is the protagonist (if you want only the protagonist to have adventures).

When this plot thread gets started, the plot agent's goal manager will use the `CrashShipWithCrew`-event to fulfill the preconditions, thus crashing the ship.

Now some additional actions and framing operators need to be added to give the character a chance to leave the island; for example, a `RepairShip` action that requires a toolbox, some trees, and a damaged ship that is moored somewhere in order to repair the ship, and of course the framing operators needed for said toolbox and trees.

Note that this coincidentally does not get in the way of the `EscapeWithTreasure`-goal; the `RepairShip` action requires the ship to be moored at a location, while still destroyed, while `MakeStuckOnIsland` required the ships to be not moored at a location, so a character that is stuck that way can't repair his ship. If

[2]The character is thrown off the ship to make the requirements of the plot thread simpler; the thread's first requirement could also state that the character was on a destroyed ship moored at the island.

this was written in a way that the `RepairShip` would get in the way, framing the toolbox (not being able to find a toolbox is more believable than not being able to find trees, or forgetting how to repair a ship) might, for example, be a framing operator that can only be used during the rising action, a climactic framing operator, or just an operator that can only be used by the protagonist, so that the antagonist still can't use it to escape. The specific option chosen can just be based on the author's preference.

Now that the first secondary conflict can be introduced, as well as a way to solve it, we add an additional conflict, where the bloodthirsty monkeys try to eat a pirate.

First we write the `EatCharacter`-goal, that has the precondition that the character with that goal is blood-thirsty, that there is a cookingpot[3] somewhere. The success condition of this goal is that a character, that isn't the character with the goal, is put into the cookingpot (symbolizing him being eaten).

Then, there is the additional plot thread, `MonkeyBusinessAdventure`, marked as a secondary conflict, with the following requirements:

- The victim is a character.
- The chaser is a character that is blood-thirsty. (The blood-thirsty part isn't necessarily required, as the precondition in the goal takes care of this. Placing it in the plot thread as well makes it easier to get an overview of the thread.)
- There is a cookingpot.
- The victim, chaser, and cookingpot are all on Dinky Island. (Or, alternatively, on the same island.)
- The chaser has the `EatCharacter`-goal.
- The victim has a goal to not be on the island with the chaser[4].

The cookingpot can either already be authored to be on Dinky Island, or a framing operator can be written for it. A group of monkeys (seen as a single character) on Dinky Island can be authored to already be blood-thirsty, or a framing operator can be used to make monkeys blood-thirsty. Some actions are needed for the monkeys to put a character in the cookingpot (and no action to leave the pot, except, for example, some sort of climactic action), that aren't described in full detail here.

Because all characters need to be on the Dinky island, this conflict can only be started after the ship wreck conflict has been started, causing it to follow it up nicely.

Now the protagonist can get into all sorts of adventures, but the antagonist needs to have something to do in the meantime. As suggested in section 4.3.3, separate plot threads can be written for the antagonist (or the same ones used), or, in this case, the `Sail`-actions can take a lot longer for the antagonist to perform than for the protagonist, giving the latter enough time for his secondary adventures to still arrive on Monkey Island for a climactic showdown.

[3]These blood-thirsty monkeys are civilized.

[4]Nicer would be a goal for the victim to not be eaten (by the monkeys), i.e. not be in the cookingpot. However, as discussed in section 4.2.3, characters can barely take other characters' actions into account when planning, so the victim would consider his goal achieved if he is not in the cooking pot at the moment.

5.3 Pirate Princess

This section shows a more complicated primary conflict, as well as how other conflicts can be added to an already existing story world (i.e. the one from the Treasure Hunt Story). The adventure concerns the rescue of the beautiful pirate princess Elaine by Guybrush out of the clutches of the evil LeChuck. Due to the more complicated nature of the primary conflict, it is introduced in pieces. Note that the world, operators, threads, goals, etc. created in the treasure hunt adventure are assumed to still exist within this story, except when noted!

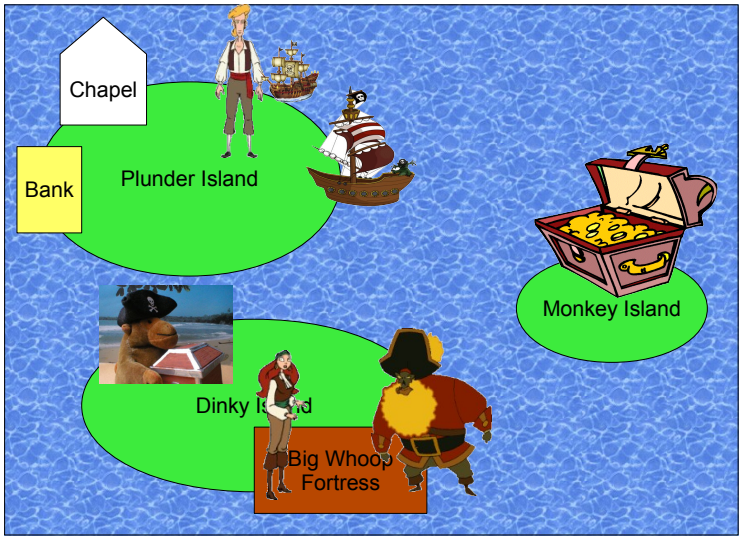

Figure 5.2: The map as it looks in the Pirate Princess Story.

The Setup The plot thread, `RescuePrincessAdventure`, which is marked as a primary conflict, has the following requirements:

- There is one character, who is both `evil` and a `pirate`, called LeChuck[5]
- There is one character, who is a `pirate` and `good`, called Guybrush.
- These is one character that is a `pirate_princess`, called Elaine.
- LeChuck is in his Big Whoop fortress (but not in its prison) at Dinky Island.
- Elaine is in the prison in Big Whoop island, and is `captive` by LeChuck.
- Guybrush is at Plunder Island.
- LeChuck carries the key to the Big Whoop prison.

[5]The names of the given characters are in reality assigned when the thread is started; Guybrush might just as well turn out to be the pirate princess, but the names are already given for readability.

- LeChuck has the goal to marry a `pirate_princess`.
- Elaine has the goal to marry a `rich` character.
- Guybrush has the goal that Guybrush has a treasure and Elaine is not `captive`.

LeChuck and Elaine have the same resolution-goal as their normal goal, because all they want to do is get married; the conflict is resolved when one of them gets married. Guybrush has a normal goal to rescue Elaine, but the resolution goal to marry her. This is because the climax occurs when a normal goal has been reached, and we want the climax to occur when Elaine has been freed from her prison, because it indicates that Guybrush has made some attempt in rescuing her. LeChuck can only get married after the climax (described below), and it is more fun, timing wise, when this happens while Guybrush has just sprung Elaine from jail.

Big Whoop, and its prison, need to be authored. The prison has a door, that can only be opened when you have its key.

In order to start this thread, a number of additional framing operators are needed:

- An operator that creates a key to the Big Whoop prison, if that key doesn't exist yet.
- An operator that makes a character a `pirate_princess` if she is a `pirate`.
- An operator that makes a character that is an `antagonist` `evil`. (Because an antagonist is not necessarily evil, it needs to be explicitly stated as such.)

Obviously, instead of using framing operators, their results can already be manually authored, to allow some control over, for example, who becomes the pirate princess.

Using these operators, and the plot thread, the set up for adventure is complete. However, none of the characters is actually capable of marrying anyone, because there are no actions to do so yet.

Marrying In this story, there are two main methods of marrying; consensual, and forced marriage. The latter is discussed first.

There is an action in the story, `ForceMarriage`. It is expected to only be used by LeChuck, due to its preconditions:

- The character performing the action is `evil` and not `married`.
- The character being married to is `captive`.
- The character being married to is not `married` yet.
- The current story phase is the `falling_action`. This is allows this action to only be executed after the climax. By only allowing this action after the climax, it gives Guybrush a reasonable chance to actually rescue Elaine, instead of LeChuck just marrying her in his first action, thus ending the story.

- Both characters are different. May sound unnecessary, but it needs to be stated explicitly, otherwise characters might start marrying themselves.

The duration of this action should be a couple of rounds, so that if Elaine is somehow recaptured after being rescued, Guybrush still has a chance to stop the marriage. When the action is performed, both participants are `married` to each other.

An expectation should exist that allows the `antagonist` to expect the `falling_action` to occur, in order to allow LeChuck's character agent to actually make a plan for the marriage. Note that this expectation is out-of-character; LeChuck himself isn't waiting for the falling action (because he isn't aware that he is in a story), but his character agent is. At the time of this writing. it has not been clearly defined yet whether this is allowed or not, but as long as expectations about things in-character and out-of-character are not mixed within the same expectation, it should not cause any problems.

The consensual marriage is achieved by three operators. The first, `ProposeMarriage`, has the following preconditions:

- The character performing the action is not `captive`.
- The character being proposed to is not `captive`. (This and the above requirements are not strictly necessary, as you can perfectly propose in prison, but it would look rather silly to the reader.)
- The character performing the action is not `married`.
- Both characters are at the same location.
- Both characters are different.

As a result, the character performing the action `proposed` to the other character.

The second action, `AgreeToMarriageProposal`, has the following preconditions:

- A character `proposed` to the character performing this action.
- The character performing the action is not `captive`.
- The character that proposed is not `captive`. (This and the above requirements are not strictly necessary, as you can perfectly accept a proposal in prison, but it would look rather silly to the reader.)
- The character performing the action is not `married`.
- Both characters are at the same location.
- Both characters are different.

As a result, the character performing this action `accepted` the (proposal of) other character.

Note that the characters can propose (and agree) to each other at the same time. Also, characters can propose and agree to multiple marriages, for whatever nefarious purpose, as long as they are not married yet.

To actually get married, the `GetMarried` operator exists. It has the following preconditions:

- One character `proposed` to the other.
- The other character `accepted` the first.
- Both characters are at a chapel.
- None of the characters is `captive`.
- None of the characters is `married`.
- Both characters are different.

As a result, both characters are `married` to each other. For simplicity, priests and such invoking the marriage are left out.

Two expectations, that expect the other character to `accept` a proposal, are needed to make an actual plan for this. The first is meant for Guybrush (and maybe LeChuck, should he build an entirely unexpected plan), which has the following preconditions:

- The character expecting the expectation has `proposed` to the other.
- The other character is a `pirate_princess`, not `captive`, and not `married`.
- The character expecting the expectation is `rich`. (Due to Elaine's goal of marrying a rich guy.)
- Both characters are different.

With this expectation, Guybrush expects any not captive, single, pirate princess to accept his marriage proposal, if he is rich (i.e. he assumes that pirate princesses want to marry rich guys).

The other expectation is probably used by Elaine, and has the following preconditions:

- The character expecting the expectation is a `pirate_princess`.
- The character expecting the expectation has `proposed` to the other.
- The other character is neither `captive` nor `married`.
- Both characters are different.

This does indeed mean that a pirate princess expects any single, free character to want to marry her. While this might seem kind of arrogant (it is), it happens to be true in this story.

The chapel is built (by the author) at Plunder Island.

All this results in LeChuck trying to force Elaine to marry him in the climax, while Guybrush tries to rescue her, become rich, take her to the chapel and then marry her, as shown in section 6. Note that LeChuck can technically try to marry Elaine consensually, but is expected to choose the evil path because it exists of only two operators (expecting the `falling_action` to occur, and then the `ForceMarriage` action).

Now all Guybrush needs to do is get rich and rescue Elaine!

The Rescue Capturing and rescuing Elaine is based on two inference operators. The first operator, `Rescue`, has as precondition that a character is not in the Big Whoop cell, was `captive` before, and has as a result that the character is not `captive`. The second operator, `Capture`, requires a character to be in the Big Whoop cell, with the door closed, not be `captive` and has as a result that the character is now `captive`.

Inference operators are used because they draw conclusions about the current state of the world; without `Capture`, a character would be just as stuck in the Big Whoop cell if the door was closed, but it would be more complicated to plan for this. Likewise, a character that is not in the cell is still not much of a captive without the `Rescue` operator, as she can do anything any other character can, but it is a lot harder to plan towards it.

In order to rescue Elaine, Guybrush needs to steal the key from LeChuck and open the prison. Due to her own goal of marrying a rich guy (requiring her to be in the chapel and not captive), she will leave the opened cell herself, and execute the `Rescue`-operator. LeChuck can then try to capture her again, by forcing her into the cell and executing the `Capture`-operator.

To force Elaine to move back into the cell again (or for Guybrush to move her out, if for some reason she does not do this herself), getting the keys (whether or not via climactic actions), fleeing the island, etc. a variety of options can be used, that are not discussed in detail here.

Getting Rich Unfortunately for Guybrush, rescuing Elaine does not win her heart; money does. To achieve this, we simply use the treasure from the treasure hunt adventure, with an additional action that requires a treasure to be brought to the bank on Plunder Island and results in the character getting `rich`. The treasure either already exists on Monkey Island, or a framing operator to create it does, so this part of the story is pretty much already there.

Secondary Conflicts The secondary conflicts from the treasure hunt story are still applicable. However, the ship-crash conflict actually brings him somewhat closer to the goal of rescuing Elaine (since Big Whoop is on Dinky Island), so it is a somewhat smaller conflict. The ship will still be broken though. The blood-thirsty monkeys, though, might even be more fun since they are now on an island where Guybrush actually needs to go!

Chapter 6

Evaluation

In this chapter, possible stories generated by the Virtual Storyteller from the authored content presented in chapter 5 is shown. The stories are intersected with comments regarding how the research presented in this thesis adds to specific parts of the story, after which a more general analysis regarding how the research benefits the entire story is presented.

Note that only two stories are presented. While it would theoretically be possible to generate, and thus analyze, more stories, unidentified performance issues have hindered this; for some, yet unknown reason, it takes far too long for the Virtual Storyteller to generate these stories. Therefore, there was not enough time within this research to actually generate more stories.

The stories themselves are presented in *italic*, while the comments are in a normal typeface. The story text is not the literal output of the Virtual Storyteller; the story itself is not changed (i.e. what happens), but its transcription is more descriptive than the actual output, which consists of a list of executed operators with their corresponding arguments.

6.1 Treasure Hunt Adventure

There is a world with two pirates. Both pirates are at Plunder Island. One of the pirates is Guybrush. The other is LeChuck. There is a treasure chest at Monkey Island. Both pirates want that treasure. There are two ships at Plunder Island. The story world setting is loaded, adding elements like the islands, ships, treasure and main characters to the world. Then `TreasureHuntAdventure` plot thread, describing the story's primary conflict, is started, causing the `MakePirate` and `PlaceCharacter` framing operators to be executed, as well as assigning the `GetTreasure` goal to the pirates. After the plot thread has been started, the story phase moves from `exposition` to `rising_action`.

Both pirates get on board their ships, and set sail to Monkey Island. The `SelectProtagonist` and `SelectAntagonist` operators are used on Guybrush and LeChuck respectively. Instead of using a generic `Sail`-action, Guybrush uses the `SailProtagonist`-action, which has a shorter duration than the `SailAntagonist`-action, used by LeChuck. This is so that Guybrush can get into all sorts of adventures, while preventing LeChuck from taking the treasure during that time. These sail-actions can only be used during the `rising_action`; dur-

ing the `falling_action`, a generic `Sail`-action is available to both characters.

Unfortunately, Guybrush's ship crashes on Dinky Island, wrecking the ship and throwing Guybrush off. A second plot thread, `ShipWreckAdventure`, is started by the plot thread manager. It describes a secondary conflict, and requires the protagonist to be stuck on an island with a crashed ship, which can be done by means of the `CrashShipWithCrew`-event. Guybrush is assigned the goal to leave the island. Note that LeChuck is still busy with his `SailAntagonist`-action.

Guybrush assesses the damage, and determines that the ship can be repaired. He gets his ax and starts chopping trees. Guybrush's character agent makes a plan to use the `RepairShip`-action, which requires timber and a broken ship to make a working ship. An ax is framed in Guybrush's inventory using the `CreateAx`-operator[1], as well as some trees on the island. Then, Guybrush uses the `ChopTree`-action, which turns a tree into timber.

Unfortunately, just as Guybrush is about to sail off with his repaired ship, a group of blood-thirsty monkeys attack him! They knock him unconscious, and start dragging him to their cookingpot. Because Guybrush has almost finished his plan (of repairing the ship), a new secondary conflict is started by the plot thread manager. The `MonkeyBusinessAdventure`-thread is started, which adds a group of blood-thirsty monkeys to the world (represented by a single character), as well as a cookingpot somewhere on Dinky Island. The monkeys have the goal to `EatCharacter` (which is modeled as the victim being in the cookingpot). Guybrush gets assigned the goal to not be on the island, which he already had (due to the `ShipWreckAdventure`), so nothing changes for him. The monkeys use a `KnockUnconsciousAndDragToCookingpot`-action on him, which places him in the cookingpot. The monkeys have achieved their `EatCharacter`-goal, and there is no additional resolution goal, so the conflict of the `MonkeyBusinessAdventure` is considered resolved (in favor of the monkeys).

Meanwhile, LeChuck arrives on Monkey Island. LeChuck's `SailAntagonist`-action has finished.

When things could not get any worse for Guybrush, he still tries to escape daringly from the cookingpot. He succeeds, runs to his ship, and leaves Dinky Island! Guybrush, the protagonist, cannot formulate any plan to achieve any of his goals (getting the treasure, and leaving Dinky Island). Therefore, the climax can take place; the story phase is set to `falling_action`, and no new secondary conflicts are introduced anymore. The story phase being `falling_action` enables Guybrush to use the `DaringEscapeFromCookingpot`-action, that moves a character out of a cookingpot. He then walks to his ship and sets sail, as normal. This makes Guybrush achieve his goal to leave Dinky Island, causing the `ShipWreckAdventure`-thread to be resolved (as it also has no additional resolution goal). The `MonkeyBusinessAdventure` was already resolved, otherwise it would be now due to this goal being achieved.

LeChuck leaves his ship, and walks to the treasure.

[1]This framing operator can only be used during the rising action; this does not become relevant in this story, but it would if, during the falling action, the antagonist got stuck on the island because every ship at it was blown up. Because he would not be able to frame an ax, he cannot repair it, and would therefore really be stuck. This works in this specific story because the antagonist would never need an ax before the rising action, as he is too busy sailing around; if that was not the case, the ax creation can for example be limited to just the protagonist.

Guybrush arrives on Monkey Island and leaves his ship.

LeChuck picks up the treasure! When LeChuck has picked up the treasure, he has achieved his `GetTreasure`-goal. He gets assigned his `TreasureHuntAdventure` resolution goal, `EscapeWithTreasure`, causing him to want Guybrush to be `stuck`, while LeChuck has the treasure and is at Plunder Island. Achieving this goal would also cause the climax to occur (since it is the goal of the primary conflict) if this had not already happened due to Guybrush being stuck in a cookingpot.

Guybrush walks to the place LeChuck is. Meanwhile, LeChuck walks to the shore, where his boat is, causing both characters to miss each other. LeChuck is planning to blow up one of the boats, while sailing away with the other, causing Guybrush to be `stuck` on the island. He has requested a framing operator that creates a cannon in the hold of a ship (in this case, his ship). Meanwhile, Guybrush is planning to steal the treasure from LeChuck, so he needs to be in the same location.

Guybrush walks to the shore, while LeChuck enters his boat.

Guybrush enters LeChuck's boat. LeChuck walks to the hold of the boat. LeChuck's plan has changed at this point; because Guybrush is not on the island anymore, he cannot be made stuck on it. Instead, LeChuck is now planning to blow up his own boat, with Guybrush on it, while he himself has left it. Since he still needs a cannon to do this, he keeps on walking to the hold.

Guybrush walks to the hold. LeChuck picks up the cannon in the hold. Cannons can easily be held in this world.

Guybrush steals the treasure. LeChuck moves to the ship's deck. This seems somewhat stupid; this is because LeChuck has planned the action to move to the deck while he still had the treasure. Because he lost it in the same round as he performed his movement, he has not adjusted his plan yet. Meanwhile, Guybrush has achieved his `GetTreasure`-goal, causing the `EscapeWithTreasure`-goal to be assigned to him. Since LeChuck is on a ship, he plans to blow it up and make LeChuck stuck that way. To this end, Guybrush used the climactic framing operator `CreateAbandonedCannon` that places a loaded cannon at the beach. Theoretically, Guybrush could have framed an additional cannon in LeChuck's ship's hold and used that one. However, because it is unwise to fire a cannon on the ship you are currently in, especially if you plan to sail to another island, he would have needed to pick up that cannon and put it down again at the beach, requiring additional actions, and therefore a longer plan.

Guybrush moves to the ship's deck. LeChuck moves to the ship's hold. LeChuck wants to steal his treasure back, and follows Guybrush. Guybrush wants to leave this ship, and therefore moves to the deck. Because this movement occurs at the same time, they switch places. In this case, it was rather lucky that Guybrush planned to use the cannon from the beach, instead of the one carried by LeChuck. If he wanted to use that cannon, they would keep on switching places between the deck and the hold forever. Stealing the cannon from LeChuck does require an additional step in the plan, causing it to work out fine in this particular situation.

Guybrush leaves the ship. LeChuck moves to the ship's deck.

Guybrush finds the abandoned, loaded cannon at the beach. He quickly uses it to blow up LeChuck's ship! Guybrush fires the cannon that he framed at the beach. LeChuck was trying to leave his ship, but Guybrush's action occurs earlier, letting LeChuck's action to leave the ship fail. Because LeChuck is now

on a destroyed ship, Guybrush can perform the `MakeStuckOnShip`-inference operator on him. Now, Guybrush has achieved two of the three subgoals of `EscapeWithTreasure`, namely him having the treasure, and LeChuck being `stuck`. All that is left for him to do is take his own ship and sail back to Plunder Island. Meanwhile, LeChuck cannot form any plan, because he is on a destroyed, and therefore non-functional, boat. Note that characters cannot leave destroyed boats; this was written precisely to facilitate this scenario.

Guybrush enters his own ship, and sails to Plunder Island. Guybrush has achieved his `EscapeWithTreasure`-resolution goal. Because one of the resolution goals of `TreasureHuntAdventure` has been met, that conflict is considered resolved. Since it was the only still running conflict, the story phase is moved to `denouement`. It would have been nice for Guybrush to perform an `EnjoyTheTreasure`-action, but since he has no real motivation to do so (as explained in section 4.4.3), so the story ends here.

Analysis As can be seen in this story, a form of dramatic structure is created. During the exposition, the characters are added to the story, and the main conflict is introduced. Then, during the rising action, things keep on getting more complicated for poor Guybrush; not only does he suffer a ship wreck that strands him on the wrong island, he also ends up in a blood-thirsty monkey's cookingpot! Fortunately, the climax takes place, and Guybrush's fortune finally smiles upon him; he manages to escape the cookingpot, and arrive on Monkey Island, to face off with LeChuck. The mechanism of adding secondary conflicts to the story really seems to work here, as Guybrush gets into all sorts of obstacles in finding the treasure, even going so far as placing him in a seemingly hopeless situation. Also, the climactic actions work here; the daring escape gets Guybrush out of the situation, while still being believable.

During the falling action, the conflict between LeChuck and Guybrush gets resolved in an explosive way: with cannons! Guybrush gets a slight advantage by being able to frame a loaded cannon at the beach (i.e. at a location that suits his plans), which causes him to win the entire conflict, while he started the falling action in a disadvantage (because LeChuck already had the treasure). However, in this case it was rather lucky that the `CreateLoadedCannon`-operator worked out; if LeChuck did not have easy access to the cannons on his ship, or would have just left the beach when it was framed there, it would be harder to believe that he would just ignore the cannon on the beach. Also, if the cannon was framed at that place while LeChuck was there, he would probably have used it, making Guybrush's use of the operator disadvantageous to him.

The falling action also shows another current shortcoming of the Virtual Storyteller; both characters are following each other for a part of the story, with nothing really happening at that time. The story that results is still rather fun though (at least in my opinion).

Once the main conflict is resolved, Guybrush sails home. The current lack of dénouement really shows here, as the story feels unfinished without it.

Parts of this dramatic structure are caused by the techniques presented in this research, while other things work out due to tweaking the authored parts specifically towards this story. The way conflicts are introduced will also work in other stories, though the author should obviously pay attention that the conflicts still make sense in that context; crashing a ship on Dinky Island will not work in

a world without Dinky Island. Likewise, the use of climactic operators will work in other stories, but the author should pay attention that the specific climactic operators still make sense in those stories. The use of resolution goals to resolve conflicts is also not directly tied to this story, and will work in other stories.

On the other hand, there are elements that are authored to work in this specific story, and can easily be disrupted if other elements are changed, or disruptive if they are changed themselves. Besides obvious elements like the laws of physics (e.g., if characters are suddenly capable of walking over water, the story would not work), and some not making sense anymore in a changed context (e.g., `DaringEscapeFromCookingpot` would make no sense without the cookingpot), some of the operators are tied to exactly this story. The most glaring example is the duration of the `SailAntagonist`-action; this duration is tied to the amount of time one expects the protagonist to require to finish off his secondary conflicts, and arrive at Monkey Island. Increasing the duration would cause the antagonist to not offer much of an opposition, since their would be no interaction between the main characters. Decreasing the duration, introducing additional secondary conflicts, or complicating the current ones, would allow the antagonist to win without much opposition[2]. Similarly, as has already been noted, the constraints on the `CreateAx`-operator work in this story because the antagonist will never need an ax before the climax, which makes sure that he cannot create one when he does need it. Also, the `MakeStuckOnShip`-operator works under the assumption that a character really is stuck when on a destroyed ship; give him a means to escape, and the story would not work anymore.

As has been noted, due to performance issues, this is the only story that has been generated with these settings and operators. Therefore, it is not really known whether a different story can result with the same inputs; while the characters can theoretically make different plans with the operators available to them, they will still try to follow the shortest plan they can make, and it is not really certain whether there are multiple shortest plans available.

6.2 Pirate Princess Adventure

The plans needed for the characters in the pirate princess adventure are longer and more complicated than those in the treasure hunt adventure, which, combined with the aforementioned performance issues, meant that it would take very long to generate a story with this input. Therefore, the story has been somewhat simplified from the original description in chapter 5. The largest change is that Elaine just wishes to marry Guybrush, instead of any rich person, thus dropping the entire treasure plot. Also, the secondary conflicts with the crashing ship and the bloodthirsty monkeys have been removed. Some smaller shortcuts have also been taken, and are noted in the story description.

There have been only a couple of test runs while authoring this story before the one shown below was transcribed. Therefore, a lot of potential improvements to the story have shown up. Instead of adding these to the story and running a new simulation, they are mentioned here explicitly in order to show both how

[2]This is partially remedied because when the treasure is picked up by the antagonist, the climax is started, allowing the protagonist climactic actions that can help him in still offering some opposition. However, these actions still need to be specifically written for this occurrence.

stories from the Virtual Storyteller can be improved by iterative process (i.e., actually running the simulation while writing the story results in better stories than just writing everything beforehand), as well as to identify less obvious issues that one needs to take into account while authoring a story.

There is a world with two pirates, and a pirate princess. The evil pirate LeChuck has taken Elaine prisoner in his Big Whoop Fortress on Dinky Island, while the good pirate Guybrush wants to rescue her. LeChuck wants to marry Elaine, while Elaine wants to marry Guybrush. Guybrush is on Plunder Island, where a ship is located, and wants to rescue Elaine. LeChuck has a key to Elaine's prison. Also, there is a chapel at Plunder Island. The story world settings are loaded, which again loads the islands and ships, as well as the characters and Big Whoop Fortress. The `RescuePrincessAdventure`-plot thread is started, which uses the `PlaceCharacter`-framing operator to place the characters in their respective locations, makes Elaine `captive` with the `Capture`-inference operator, gives LeChuck the key to the prison, and gives the characters the `MarryPerson` and `RescuePerson`-goals. The other properties of the characters (their pirate-ness, whether they're protagonist or antagonist, etc.) could have been filled in with framing operators, but due to the performance issues, the length of the plans that the planner can make has been limited. A plan to fill in all these properties with framing operators would have created a plan that was too large to fit, so these properties were added to the setting information.

Elaine sits in her prison, hoping to be rescued. Elaine needs to go to the chapel on Plunder Island to get married, but is stuck in prison. In the story as described in chapter 5, she would not be capable of making a plan, which would probably make her lose hope if she had some emotional model, and would therefore be the preferred method. However, because it takes really long to find out that a character is not capable of making a plan, an expectation-operator has been added that makes her expect the door to be opened. So, until that actually happens, she is just in prison, expecting to be rescued.

LeChuck is doing nothing. In reality, he is expecting the falling action to occur, as this is a requirement for the `ForceMarriage`-action that he wishes to take. It would be nice if he would be performing some generic `PrepareForWedding`-action that has no real effect in the meantime, for the benefit of the reader. While it is not yet possible to have him do that the entire time, it could have been made a requirement for the `ForceMarriage`-action to have prepared the wedding, so that the `PrepareForWedding`-action has at least been executed once.

Guybrush goes aboard his ship, and sets sail to Dinky Island. Once he arrives there, he leaves his ship and enters Big Whoop Fortress. Guybrush just follows his plan. This would have provided a couple of good opportunities for secondary conflicts; not only could his sea-voyage have been made more difficult, the method of entering the evil fortress (i.e., he just walks in) was rather lame while it could have provided lots of excitement. This would have made the story more complicated, presumably resulting in the aforementioned planner performance issues, though.

Inside the fortress, Guybrush encounters LeChuck. Guybrush steals LeChuck's key. This clearly shows the problem with characters not considering that the

other might be up to something. Since LeChuck does not know that Guybrush is stealing the key to free Elaine, he does not react to it in any way. This could have been partially solved by adding having the key to the preconditions of the `ForceMarriage`-action, thus causing LeChuck to do something about losing it, though some method in which he just wants to get anything stolen from him back would be a better solution.

Guybrush opens the prison's door with the key. Still, LeChuck does not see this as harmful to his plans. After opening the door, Guybrush expects Elaine to leave by herself.

Elaine leaves the prison. With Elaine's expectation that the prison door would open fulfilled, she can continue with the rest of her plan to get married, the first step of which is to leave the prison.

Then, things happen internally, that the reader of the story will not see. First, Guybrush performs the `Rescue`-inference operator, that removes Elaine's `captive`-status. Note that Elaine requests the same inference operator, but Guybrush just happens to be first. Then, with Guybrush's `RescuePerson`-goal having been fulfilled due to Elaine not being `captive`, a goal in the primary conflict has been fulfilled. This starts the climax (i.e., sets the `story_phase` to `falling_action`), and gives Guybrush his resolution goal, `MarryPerson`, with Elaine as the person he wants to marry. Meanwhile, Elaine not being `captive` anymore breaks a precondition of LeChuck's `ForceMarriage`-action, so he finally starts doing something.

Guybrush proposes to Elaine. Proposing does fulfill one of the requirements of consensual marriage. Again, due to characters not really being aware of other characters, LeChuck being in the same room does not persuade Guybrush to wait for a more appropriate time. This could have been solved by adding a requirement to the propose action that it should not take place on Dinky Island.

LeChuck knocks Elaine unconscious. When a character is unconscious, she can be dragged around by other characters. In this case, LeChuck wishes to drag her back to the prison. Note that allowing characters to become unconscious requires all actions to have a precondition stating that a character should be conscious.

Elaine starts recovering. When unconscious, characters can perform a `Recover-Consciousness`-action that makes a character conscious again. It takes three rounds to perform though.

Guybrush scoops up Elaine and flees Big Whoop Fortress with her. Guybrush performs the climactic `DaringRescueAndEscape`-action, that allows him to move himself, and any person in the same location as him, to an adjacent location.

LeChuck tried to pick up the unconscious Elaine, but she was already snatched away by Guybrush. Since LeChuck tried to pick up Elaine in the same round as Guybrush performed his daring move, he has not been able to react to it yet.

Guybrush throws the unconscious Elaine aboard his ship and boards it. Guybrush performs the `DaringRescueAndShipBoarding`-action, which is a companion to the `DaringRescueAndEscape`-action, but different because boarding ships uses a different internal mechanism than moving between adjacent locations.

LeChuck leaves Big Whoop Fortress while chasing Guybrush and Elaine. LeChuck just moves outside of Big Whoop Fortress, in the hopes of still capturing Elaine.

Guybrush sets sail with his ship.

LeChuck breaks down in failure. Well, not really, but he is just incapable of making any plans to capture Elaine because he does not have any ships himself. It would have been nice to add this, because it makes the chase a bit more spectacular and allows it to end in ship to ship combat, which is always awesome. The rest of the story, he is not doing anything.

Guybrush arrives with his ship at Plunder Island.

Elaine recovers from her unconsciousness. Her `RecoverConsciousness`-action finished.

Guybrush leaves the ship. Guybrush expects that Elaine will accept his proposal, so he rushes ahead to fulfill the remaining condition of the `GetMarried`-action, namely being at the chapel. It might have been a good idea to add 'being at the same location' as a precondition to that expectation.

Elaine leaves the ship. Elaine is trying to be at the same location as Guybrush, in order to accept his earlier proposal.

Guybrush enters the chapel.

Elaine enters the chapel.

Guybrush waits. All preconditions, except for the accepted proposal, have been fulfilled. He is now just waiting for that acceptance, which he still expects to occur.

Elaine accepts Guybrush's proposal.

Guybrush and Elaine get married! Drinks all around! Guybrush performs the `GetMarried`-action, that makes Elaine and Guybrush married to each other. Elaine also tries to perform that operator, but Guybrush was first. Since one of the resolution goals in the primary conflict has been achieved (and there are no further conflicts running), the story is finished.

Analysis Due to this story needing to be simplified, it does not really do a very good job of showing how the techniques presented in this thesis can add to a story; while it does show how the primary conflict of a story can be introduced by means of a plot thread, and it included two climactic actions, it does not really show anything in this regard that the treasure hunt story did not do, so this is not discussed in detail.

On the other hand, this story does illustrate two remaining issues; the performance issue, and the issue that characters do not really consider each other.

Due to the performance issue, the length of a character's plan is rather limited. This not only caused the main story to be simplified, but also made it impossible to introduce secondary conflicts into the story that made Guybrush's rescue attempt more complicated; this would add more operators to his rescue plan (in order to work around those complications), causing it to exceed the maximum plan depth. While it is theoretically possible to just increase this depth, it increases the time to make a plan exponentially, making it rather impractical. At the time of this writing, it is still not clear what causes this low performance.

The second issue, of characters not really considering each other, was covered up in the treasure hunt adventure, by making the other character an integral part of the goals of the characters. This was not done in this story, causing weird situations like Guybrush just freeing Elaine, and proposing to her, under LeChuck's nose. Also, while it did seem like Guybrush was fleeing with Elaine from LeChuck, in reality he was just using his daring moves because they were

the quickest way to achieve his marriage goal. Guybrush seeing LeChuck as a threat to his plan, and vice versa, would have made them react to each other, and would have made Guybrush's rescue attempt somewhat more exciting, and would have made the climax more interesting.

Chapter 7

Conclusion

In this research, an attempt has been made to answer the research question: 'How can we introduce the structure of Freytag's pyramid in emergent narrative while still maintaining the autonomy of the characters?' In section 2.4, this has been split into six subquestions:

1. How can a conflict be introduced?
2. How can the resolution of conflicts be facilitated?
3. When should conflicts be introduced?
4. What makes a conflict a secondary conflict?
5. When is the right time to resolve a conflict?
6. When does a story end?

This chapter will first summarize how these subquestions have been answered, after which it will be discussed how successfully the main research question is addressed by them.

Conflicts are introduced by plot threads. These plot threads are schemas that describe the premise of a conflict; all elements that are necessary for a conflict are stated as requirements on the world. For example, a plot thread about a treasure hunt will state that there need to be two pirates, a treasure, and that both pirates want the same treasure. The plot agent then tries to fulfill these requirements by using framing operators and events to fill in the world. Once a plot thread has been started (i.e., all requirements have been fulfilled), it does not directly steer the story; while it does describe how a conflict should end in a general sense (e.g., one of the pirates has the treasure), it does not contain any specifics how this end should be reached. This way, character autonomy is largely maintained; characters need to find out for themselves how their goals can best be achieved, while forming the story around them.

Once a conflict has been introduced, it needs to be resolved in some way. While it is theoretically possible for the characters to do this without any outside interference by the plot agent, it usually does not happen. Two causes for this have been identified; the characters' goals might not resolve the conflict, and the characters may not be able to resolve it.

The goals not resolving the conflict is usually because a character's real goal does not concern the other character. Thus, if a character has achieved his goal, there is still the possibility of the other character interfering with it. Therefore, a conflict can only be seen as resolved when it is clear that one character has achieved his goal, and the other character is not capable of interfering anymore. For example, a character's real goal is to get a treasure, and he does not really care about what happens to the other character. However, this conflict is not resolved by just having the treasure, because the other character can then still steal it. In fact, this scenario will most likely end with both characters standing next to each other, taking turns in stealing the treasure from the other character. To this end, an additional resolution goal can be assigned to a plot thread, describing an additional goal for characters that does resolve the conflict. This resolution goal is given to a character as soon as he has achieved the other goal. In the case of the treasure hunt, it can describe that the character with the resolution goal should be with the treasure at home, while the other character should be stuck on an island.

Characters not being capable of resolving a conflict is usually caused by both characters having the same abilities; no character has an advantage over the other, so the conflict results in a stalemate. This is solved by actually giving the protagonist an advantage; after the climax, the protagonist can access additional actions, events and framing operators, that give him a slight advantage. Examples of these climactic operators include 'Win a swordfight', 'Have a lucky cave in seal the antagonist away' and 'Create a cannon in an unlikely place'. This gives the protagonist the means to resolve a conflict in his favor, thus making sure it gets resolved at all.

Besides the primary conflict of the story, that is introduced at its start, additional secondary conflicts get introduced during the rising action, to make the story more exciting and add suspense. However, the timing of this is a bit tricky; introduce too many secondary conflicts after each other, and the story can get very convoluted, or even tiresome. Introduce too few, and the protagonist seems to be having a walk in the park. Unfortunately, no real solution could be found to detecting when a new conflict should be introduced. Instead, a couple of simple guidelines have been introduced:

1. As soon as the story is started.
2. When the protagonist has not taken any action for a number of rounds.
3. When the protagonist has almost finished his currently active goal.
4. If, at any time, no secondary conflict has been running for a number of rounds.

A human author can select which guidelines will be adhered to in the simulation of his story.

Due to the performance problems in generating stories, it was not possible to evaluate them exhaustively; within the Treasure Hunt Adventure (section 6.1) the guidelines seemed to work well, but it is not possible to say with certainty that they will work as well in other stories.

The difference between primary and secondary conflicts has been discussed, with the conclusion drawn that there is no intrinsic difference between them, just the context in which they are used; what is a secondary conflict in one story can be the primary conflict of a different story, and vice versa. Therefore, the author of the plot threads that describe the conflicts is given control over whether that thread will be used as a primary or secondary conflict.

Following Freytag's pyramid, all unresolved conflicts should be resolved after the climax. It has already been answered how these conflicts can be resolved, so this question boils down to: When should the climax occur? Since the climax marks the protagonist's reversal of fortune, it should preferably occur when he is incapable of making a plan for any of his goals; his situation seems hopeless, and only incredible luck or feats of awesomeness (i.e. the climactic operators) can save him! However, it can occur that this never happens, and the protagonist can always find some plan. To that end, two additional heuristics are added; as soon as the goal of the primary conflict has been met, or after a certain number of secondary conflicts have been introduced, the climax occurs.

A story ends when all conflicts have been resolved. After this, the dénouement takes place, in which characters can perform celebratory actions to indicate their success in resolving the story's conflicts.

Evaluation Now, how do the answers to these subquestions address the main research question, 'How can we introduce the structure of Freytag's pyramid in emergent narrative while still maintaining the autonomy of the characters?' As shown in chapter 6, applying the results of these subquestions to the Virtual Storyteller creates a system that is capable of generating stories that follow Freytag's dramatic structure within emergent narrative. However, it only gives the author the tools to do so, most notably allowing the introduction of conflict within the story, by means of plot threads, as well as having the current story phase stored as knowledge, thus allowing or disallowing certain operators at certain times. It does not enforce the dramatic structure if the author does not use these tools.

Whether the methods used in this thesis also work in other emergent narrative systems than just the Virtual Storyteller mainly depends on two things: the system needs to be capable of introducing new parts to the world while within the story, and the characters may need to accept goal suggestions.

If no new parts of the world can be introduced while the story is being generated, it becomes (nearly) impossible to add secondary conflicts. In that case, all elements needed for the secondary conflicts need to be introduced at the start of the story, which means that either the specific secondary conflicts that are going to take place within that story need to be known beforehand, or the required parts of the world for all authored secondary conflicts need to

be introduced already, which both clutters the world and prevents secondary conflicts that are contradictory when both are started from being written.

The characters not accepting goal suggestions may or may not be a problem, depending on how the characters in other emergent narrative systems do take on goals. The following three reasons a character receives a goal suggestion can be differentiated within this thesis:

- As the goal for the primary conflict. This goal is suggested at the start of the story, when nothing has happened yet. It can be seen as part of the identity of the character; like stating that a character is a pirate called Guybrush, one can state that his goal in life (or, at least, in the story) is to get a treasure. Since most emergent narrative systems give the author at least this much control at the start of the story, the primary conflict can be introduced this way in most other systems.

- As a goal for a secondary conflict. A goal suggestion received due to a secondary conflict being introduced is different than one from the primary conflict, because the story has been running for a while; the character has already been doing things, and needing to accept a new goal can be seen as an intrusion on their autonomy, and might therefore not be possible in every emergent narrative system. However, the goals in most secondary conflicts will presumably make sense to assume anyway, if the character involved was capable of drawing the correct conclusion about his predicament; if a character's ship crashes, he will probably assume a goal to repair it anyway, whether this was suggested due to the secondary conflict being introduced or not. Likewise, if a horde of blood-thirsty monkeys is chasing him, he should try to escape them. Therefore, if the characters in an emergent narrative system do not accept goal suggestions while the story is running, but are capable of the reasoning required to take on these goals anyway, the method of introducing secondary conflicts as presented in this thesis is still viable.

- As a resolution-goal. As with goals from secondary conflicts, resolution-goals are also suggested while the story is already running, and can therefore be seen as an intrusion on a character's autonomy. Similarly, resolution-goals are goals that will probably be taken on by a character in that situation anyway, if that character was capable of reasoning enough about their situation; as has been stated in section 4.2.2, if a character has a treasure, and knows that another character is hunting for the same treasure, it makes sense that he will assume the goal to put the treasure in safety somewhere. In this thesis, the resolution-goals are suggested to a character precisely because they are not capable (yet) of drawing these conclusions (regarding which goal to assume) in the Virtual Storyteller. If, in another emergent narrative system, the characters are capable of doing so, they would not need to accept goal suggestions.

In short, while characters need to accept suggested goals at the start of the story, the suggestions characters currently receive during the story are not necessary if the characters are sufficiently capable of reasoning about the situation they're in.

This also brings us to the last part of the research question: How far *is* the characters' autonomy maintained? In the specific implementation as presented here, the characters are still capable of choosing how they achieve their goals, but not in choosing what these goals are. However, as discussed above, the goals that are assigned to characters are (or, at least, should be) in most cases goals the characters probably would have chosen anyway, if they would be capable of doing so, and assigning them directly is a Virtual Storyteller specific workaround. The main method of influencing characters into following the dramatic structure is still by modifying their environment, and the goals that currently get assigned to characters should make sense due to the changes in the environment. Therefore, while the characters' autonomy is currently negatively impacted by this, the method of introducing conflict presented here should be possible to use in the future without influencing that autonomy.

Another issue regarding the characters' autonomy is that operators can be allowed or disallowed during a story, depending on the character that wants to perform it, and the current dramatic phase. This can be seen as impeding a character's autonomy; that character is suddenly forbidden from doing something that other characters still can, or that he can do at another time, without a real in-story explanation. On the other hand, this can also be seen as a character being capable of doing things other characters cannot, or as being capable of doing them better; the character is still capable of making his own decision (he can choose not to perform that operator), and therefore autonomous, and is just a bit more awesome than the other characters. Whether this intrudes on character autonomy is a matter of personal opinion.

The main research question has been answered; the method presented in this thesis is capable of introducing dramatic structure into the Virtual Storyteller, and (assuming some conditions are met) emergent narrative in particular, while reasonably maintaining the characters' autonomy. But, does this automatically lead to more interesting stories?

While the introduction of conflicts, and the following dramatic structure, does have the potential to lead to interesting stories, it does not necessarily do so; conflicts that get resolved in a single round do still technically count as conflicts, even though they barely contribute anything to the sense of suspense and excitement in the story. Similarly, while the conflict might actually be running for quite a while, it still is not really interesting if nothing happens during that time; e.g., Guybrush and LeChuck might be working to achieve their goal, while not interacting with each other in the least. As shown in chapters 5 and 6, these problems can partially be worked around with some effort of the story's author, but the methods to do so are not universally applicable; what works with one story might have the complete opposite effect in a different story. Most of these issues, like characters not taking each other into account when planning, characters knowing the exact effect of every action they're taking, have already been identified in chapter 4, and are repeated and expanded upon in chapter 8. Since they are rather numerous, they are not all mentioned in this conclusion.

Taking this into account, this research's main contribution to the Virtual Storyteller, and story generation in general, is not necessarily a system that adds dramatic structure to an emergent story, but more a framework that can

be expanded upon in order to allow this, as well as identifying the possible research directions to do so.

It is my hope that my thesis has inspired future researchers to work in this direction on the Virtual Storyteller, as I am really curious about the stories that can follow from it, whether they are piratey or not.

Chapter 8

Future Work

While the system presented in this thesis does introduce some semblance of dramatic structure into the stories generated with the Virtual Storyteller, a lot of work needs to be done to make really interesting stories. Points for this future work are given in this section. Most points have already been addressed in chapter 4, but they are gathered here for convenience. Other points have been briefly mentioned already, but are expanded here.

Note that this section only gives points for further research; things that have not been implemented due to time or dependence on other research but are not research subjects on their own are not mentioned.

Goal Acceptance As has been mentioned numerous times, characters can refuse to accept a goal, which, for example, may cause a conflict plot thread to be not conflicting at all; for example, if only one of the two pirates accepts the goal to get the treasure, the story will not be that exciting. Similarly, if the pirate accepts the goal to flee from the blood-thirsty monkeys, but the monkeys do not accept the goal to eat the pirate, the story will be rather lame. Vice versa, if the blood-thirsty monkeys do accept the goal to eat the pirate, but the pirate has no interest in fleeing from them, the story will become rather strange.

Some hypothetical reasons why a character might not wish to accept a goal are that it does not conform with their ethical values (e.g., pacifists might not wish to kill someone, vegetarian bloodthirsty monkeys will only eat blood oranges, not pirates), they might be too busy with another goal (e.g., LeChuck is already conquering the world, so he does not care to hunt for a comparatively insignificant treasure), or they might just not be interested in the goal (e.g., perhaps Guybrush really does not want to marry the arrogant pirate princess).

Since there is no model yet regarding whether the characters either accept or refuse these goals, there is no clear way to work around this. At the moment, this is solved by some general handwaving (i.e. the goals stated in a plot thread should make sense within the context specified in that plot thread, and will therefore probably be accepted), but this should be stricter formulated. Possibilities include that the characters try to find some way to justify the suggested goal (e.g., LeChuck might find some way to incorporate that insignificant treasure in his world dominating plans, Guybrush suddenly thinks of the enormous treasure owned by the pirate princess, giving a new view to the marriage), and the plot agent modifying the goals, or the world, to make them fit (e.g., while the

vegetarian blood-thirsty monkey might not want to eat the pirate, he will still try to capture the pirate because he is carrying those delicious blood oranges). However, there will still be situations were the goal just cannot be accepted, and a better method[1] might be needed to handle those cases.

Goal Adoption Related to the goal acceptance issue is the goal adoption issue; characters in the Virtual Storyteller are capable of adopting goals on their own. This theoretically allows sidestepping the entire goal acceptance issue; if characters know what goal would make most sense to adopt in every part of the story, it would not be necessary anymore to directly suggest it. For example, if Guybrush would know that being on an island with blood-thirsty monkeys is detrimental to his health, he could just adopt the goal to flee the island himself, instead of it being suggested by the plot thread. Likewise, knowing that LeChuck is coming to steal Guybrush's treasure can allow him to adopt the goal of making LeChuck stuck somewhere, instead of it needing to be suggested as a resolution-goal.

This issue is largely related to characters not really knowing that they are in conflict with each other, discussed further below. While it can probably be partially solved by adding preconditions to the goals (e.g., the character should be on an island with blood-thirsty monkeys), and then adopting every goal of which the preconditions are met, it still requires dealing with urgency (fleeing from the monkeys is more important than burying the treasure), and a more general solution to reasoning about these goals would be nice.

Cooperative Conflict Resolution As briefly mentioned in section 4.2.2, the sole responsibility of solving a conflict lies on the protagonist. However, the antagonist can theoretically also participate, by failing at actions (e.g. shoot with a cannon, but miss) or by choosing a plan that is not the ideal plan, but does make things somewhat easier for the protagonist. This requires two main areas of research: separation of character and actor, and failing believably.

The separation between character and actor is needed because helping the protagonist is definitely not in the best interest of the antagonist (i.e. the character), but might be in the best interest of the story (and therefore, the actor). This requires the character agent to know when an action benefits the character, and when it benefits the story, which is definitely not a trivial problem.

The second area of research, failing believably, is needed to make it believable that the antagonist does not act in its own best interest. This is not that difficult in the case of non-deterministic actions (e.g. shooting a cannon and missing), but becomes a lot more complex if the antagonist needs to choose a plan that is not ideal. The latter indicates that either the antagonist does not have perfect knowledge (which raises the question what he can realistically *not* know) or just made a mistake while planning (which raises the question which mistakes can be made without making the antagonist look like an idiot). In the first case, LeChuck not knowing that Guybrush has access to ancient voodoo magic that

[1] Currently, the goal is given to another character that does accept it. If none are available, the plot thread the goal originated from is dropped. While this is not necessarily a problem when starting a plot thread, it does present problems when it is a resolution-goal that is being suggested.

can banish LeChuck from the world of the living is acceptable, but LeChuck not knowing that Guybrush is capable of opening doors is rather silly. In the second case, LeChuck taking a route to the treasure that is slightly longer (e.g. walking east and then north instead of directly northeast) is acceptable as an non-ideal plan, but LeChuck swimming from Plunder Island to Monkey Island instead of taking the boat is not.

This area of research is explored in Riedl and Stern [2006], though it is not directly related to emergent narrative. The major plot points that need to occur in a story (e.g., in a story about a terrorist attack, it is required for a bomb to go of) are layed out beforehand. To each of these plot points, preconditions are attached that need to be true in order for the plot point to take place. Actions that can theoretically disturb such a precondition also have an outcome that will fail in disturbing that precondition (e.g., shooting an important character has a possible outcome that the gun jams, and the character therefore remains alive). If an action is performed that would disturb a precondition (and there is no alternative way of meeting all preconditions), the failed outcome will take place.

As stated, this method requires major plot points to be known beforehand and might therefore not be immediately applicable to emergent narrative. It can be a nice starting point, though.

Fullfilling expectations of other characters, mentioned in section 4.2.3 as well as the next paragraph, also addresses these same problems.

Other Characters When characters are making plans, they do not take into account that other characters can also perform actions. This makes characters incapable of making plans that involve the other character being in a different situation than the current one, and makes the characters not see the other characters as a threat; i.e. they are not really aware that they are in conflict with each other.

The first problem is largely solved by the expectation schemas, discussed in section 4.2.3; characters can expect the other characters to do something in a given situation (e.g., accept a marriage proposal if the one doing the proposition is rich), allowing them to make plans involving the other character. However, there are two problems still left open. The first concerns how long a character can wait for his expectation to be fullfilled; if he just proposed, it would be rather silly to have him sit still and wait for an answer, while the character being proposed to goes off to have an adventure, totally ignoring the first character. On the other hand, it would be rather impatient to lay a trap, expecting the other character to walk into it, and go and do something else if the trap is not triggered within the next second. This can partially be solved by having other characters be aware of and address the expectation of the first character, by either fullfilling it (tying into the research regarding "cooperative conflict resolution", discussed in the previous paragraph), or explicitly refusing to fulfill it (e.g., saying 'No!' to the marriage proposal). The second problem with expectation schema's is when a character should choose an expectation or an action; usually, using an action makes more sense (i.e. making sure something happens, instead of just waiting and hoping it happens), though there are exceptions; if you know a character is sailing to your island, you might just as well wait for that character to arrive instead of sailing towards him.

The other problem regarding other characters is that characters do not see other characters as a threat to their plans; LeChuck standing next to the treasure is not seen as a problem in Guybrush' plan to get said treasure, because LeChuck does not have the treasure yet, and Guybrush does not take into account that LeChuck can pick it up. Likewise, LeChuck wanting to marry Elaine is not seen as a problem for Guybrush, who also wants to marry her, because LeChuck hasn't married her yet.

The problem in the treasure hunt example is partially solved by having a resolution goal that states that the other character should be stuck on the island; this makes sure that the characters try to get rid of the other character. However, they aren't really aware that they are doing this because the other character is a threat to them having the treasure instead of it 'just' being a goal they have.

In the marriage example, this is partially solved by making the starting situation so that Guybrush is actively working against a precondition of LeChuck's marriage-action in order to fulfill the preconditions of his own marriage-action (i.e. that Elaine needs to be free, see section 5.3 for more detail). As with the treasure-example, Guybrush is not aware that doing this disrupts LeChuck's marriage-plans, and does not realize that LeChuck might come after him to recapture Elaine.

Solving this on a more fundamental level, instead of on a per story basis, requires characters to either be aware of each other's plans and goals, or be capable of reasoning about them by means of the actions they have seen the other characters take. Then, this information needs to be taken into account when making their own plans, which requires a large overhaul of the character agent's planner.

Nondeterministic Actions A problem with the current actions in the Virtual Storyteller is that their outcome is known beforehand *to the characters*. While this will not hinder the conflicts per se, the flow of the story itself is affected; for example, characters will not start sword fights with each other if they know they are not going to win, greatly reducing the amount of action in a story. Similarly, if a pirate knows that drinking the poisoned rum will get him killed, there is no way he is going to drink it, limiting some methods of solving the conflicts. Even if the pirate did not know that the rum was poisonous (see "False Knowledge", further below), due to knowing the effects of each action, and therefore that drinking this particular rum will cause dead, whether due to poison or not, he will still not drink it.

This problem is split into two different subproblems; the first is that actions may not *always* have the desired outcome, and the second is that the outcome might be entirely different from what was expected.

The first problem addresses situations in which the outcome is not known beforehand, but the possibilities are known. For example, in a sword fight, it is known beforehand that you can either win or lose, but it is not known which of the two will be the actual result. Likewise, shooting a cannon can either hit or miss the target, but it is not known which of the two will be the result.

A general solution is to add multiple outcomes to an action, with a probability added to each outcome. When the action is performed, the resulting outcome is chosen randomly (taking probability into account). When placing

the action into a plan, its cost is somehow modified by the probability of the desired outcome (i.e. an action with a high probability of achieving the desired result is preferred to an action with a low probability of that result).

This solution still has a number of unresolved issues; a character might have such a dislike for one of the possible outcomes (e.g. dying in the sword fight), that he may not want to risk it. Some model is needed for this. Likewise, the plot agent can select one of the outcomes (instead of choosing it randomly) if it benefits the story.

The second problem with deterministic actions is about situations in which the results of the action (or the chances of the results of that action) are different from what is expected; for example, rubbing a lamp may cause a genie to get out, instead of (or in addition to) making the lamp shiny. Drinking a bottle or rum might kill you, because it was secretly poisoned. Likewise, a cannon may be broken, unknown to the character using it, causing its shooting to be highly inaccurate.

A solution to this might be to present two different sets of outcomes in an action; one set is what the character's planner takes into account, while the other set is what actually occurs. With the example with the lamp, the outcome given to the planner is that the lamp gets shiny, while the actual result is that the lamp gets shiny, and a genie comes out. In the cannon example, the planner might think that there is an 90% chance of hitting (and 10% of missing), while the actual outcome has a chance of 10% of hitting.

A large unresolved issue with this solution is if, and how, characters should be capable of learning the actual outcome. In the lamp example, the character might (a) learn that rubbing *this* lamp will result in a genie coming out, (b) that rubbing *any* lamp will result in a genie coming out, or (c) not learn anything about this experience at all. In the cannon example, firing one shot might not tell the character that much, but he can also learn the new probabilities, or the reason that the cannon has a different probability set (i.e. that it is broken). No solution is offered to determine what a character can learn in what situation.

Note that the incomplete solutions presented above are just suggestions to starting points regarding research in this area; they have in no way been tested, and can of course be entirely disregarded in favor of a better solution.

Timing the Secondary Conflicts While section 4.3.2 addresses the timing of introducing secondary conflicts, it does so with a rather hackish list of guidelines, in order to have at least something in place. The research regarding the timing issue is very limited; ideally, secondary conflicts should be introduced when the story starts getting dull, but further research is required to actually detect this.

The Antagonist's Adventures While the protagonist is having secondary adventures, the antagonist should not just complete his primary goal, because that would be rather lame. Section 4.3.3 discussed two solutions:

1. The antagonist goes on adventures just as well.
2. The antagonist is doing nothing in the meanwhile, assuming that the reader does not really care about it.

While it is possible to implement both solutions with the research in this thesis, some issues with them are only vaguely addressed.

When sending the antagonist on secondary adventures as well, it needs to be assured that he is still alive and capable to meet the protagonist in the story's exciting climax. The protagonist can escape from his adventures by use of his 'awesome climactic superpowers', but the antagonist does not have these. A possible solution is to give these powers to the antagonist as well, but only when used in his secondary adventures, because the protagonist would otherwise have no advantage over the antagonist. This, however, reduces believability when the antagonist does not use the superpowers he just used to escape the bloodthirsty monkeys to leave Monkey Island with the treasure. This needs to be addressed somehow, though research in cooperative conflict resolution (mentioned above) might be of help here.

If the antagonist is not doing anything at all during the protagonist's adventures, this can reduce believability (because he is not doing anything, instead of e.g. trying to get the treasure). However, because nothing is mentioned regarding the antagonist when he is not doing anything, it can be interpreted as the antagonist having all sorts of adventures himself, but that just are not told to the reader. However, this will make the antagonist less interesting; he will not be seen as that much of a menace if he is barely doing anything. This can probably be partially solved by having him do some generic bad guy stuff (e.g. `TerrorizeRandomCitizen`) that does not have much impact on the story, but the antagonist needs a reason to do this (probably requiring an emotional model), and the antagonist is not necessarily *evil*. More research is required in this area.

False Knowledge Some conflicts are impossible to script at the moment: those regarding incomplete knowledge of the character. For example, the character might not know where the treasure is, or might think that the treasure is at a different location than in reality.

At the moment, the characters have perfect knowledge of their world; they know everything. However, their beliefs about the world are modeled separately from the actual world; they just happen to match because they get perceptions about everything that happens in the world. In order to have conflicts with incomplete knowledge, the characters should only get limited perceptions. This requires some model regarding what the characters can and can't see, as well as some method to update their beliefs (e.g. viewing a treasure map might give them knowledge of the layout of an island, walking into a room gives knowledge about that room), as well as to falsify them (e.g. seeing LeChuck on Monkey Island should remove the belief that he is in his Big Whoop fortress). This model is not trivial, and therefore requires additional research.

Bibliography

Owl web ontology language overview. URL `http://www.w3.org/TR/owl-features/`.

Resource description framework. URL `http://www.w3.org/RDF/`.

Aristotle. Poetics. 1902. Translation by Butcher, S.H.

R. Aylett. Narrative in virtual environments - Towards emergent narrative. *Working notes of the Narrative Intelligence Symposium*, 1999.

R. Aylett, S. Louchart, J. Dias, A. Paiva, M. Vala, S. Woods, and L. Hall. Unscripted narrative for affectively driven characters. *IEEE Computer Graphics and Applications*, 26(3):42–52, 2006.

L.M. Barros and S.R. Musse. Introducing narrative principles into planning-based interactive storytelling. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, pages 35–42, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-110-4.

J. Campbell. *The hero with a thousand faces*. Princeton University Press NJ, 1973.

S. Faas. Virtual storyteller - an approach to computational story telling. Master's thesis, University of Twente, 2002.

R. Figueiredo, J. Dias, A. Paiva, R. Aylett, and S. Louchart. Shaping emergent narratives for a pedagogical application. *NILE-Narrative and Interactive Learning Environments*, pages 27–36, 2006.

G. Freytag. *Technique of the drama - An exposition of dramatic composition and art*. Scott, Foresman and Company, Chicago, 1900. Translated from the sixth German edition by E.J. MacEwan.

E. E. Kruizinga. Planning for character agents in automated storytelling. Master's thesis, University of Twente, 2007.

M. Mateas. An Oz-centric review of interactive drama and believable agents. *Lecture notes in computer science*, 1600/1999:297–328, 1999.

M. Mateas and A. Stern. Façade: An experiment in building a fully-realized interactive drama. *Game Developers Conference, Game Design track*, 2003.

James R. Meehan. Tale-spin, an interactive program that writes stories. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 91–98, 1977.

Microsoft Corporation, 2003. URL `http://www.microsoft.com/msagent/default.asp`.

V.I.A. Propp. *Morphology of the folktale*. University of Texas Press, 1968.

S.H.J. Rensen. De virtuele verhalenverteller: Agent-gebaseerde generatie van interessante plots. Master's thesis, University of Twente, 2004.

M.O. Riedl and A. Stern. Failing believably: Toward drama management with autonomous actors in interactive narratives. In *Technologies for Interactive Digital Storytelling and Entertainment*, pages 195–206. Springer Berlin / Heidelberg, 2006.

T.C. Smith and I.A.N.H. Witten. A planning mechanism for generating story text. *Literary and Linguistic Computing*, 2(2):119, 1987.

I. Swartjes. The plot thickens - bringing structure and meaning into automated story generation. Master's thesis, University of Twente, 2006.

I. Swartjes and M. Theune. The virtual storyteller: Story generation by simulation. In *Proceedings of the Belgian Netherlands Artificial Intelligence Conference (BNAIC'08)*, Boekelo, the Netherlands, 2008.

I. Swartjes, E. Kruizinga, and M. Theune. Let's pretend I had a sword - late commitment in emergent narrative. In *Proceedings of Interactive Digital Storytelling (IDS'08)*, Erfurt, Germany, 2008.

M. Theune, S. Rensen, R. op den Akker, D. Heylen, and A. Nijholt. Emotional characters for automatic plot creation. pages 95–100, 2004.

M. Theune, N. Slabbers, and F. Hielkema. The narrator: NLG for digital storytelling. In *Proceedings of the 11th European Workshop on Natural Language Generation (ENLG'07)*, pages 109–112, Schloss Dagstuhl, Germany, June 2007.

M. Wooldridge. *An Introduction to Multi-Agent Systems*. John Wiley & Sons, 2002. ISBN 047149691X.

Glossary

Action

An in-character operator that describes an action directly performed by a character, e.g., picking things up. *Page 20*

Character Agent

An agent that controls a character in the story world, and allows it to think about the world, and influence it. *Page 20*

Climactic Operator

Any operator that can only be performed after the climax, usually only by the protagonist. These operators give the protagonist a better chance of winning the conflict. *Page 36*

Climax

The third part of Freytag's pyramid, which forms the turning point for the protagonist's fortune; while things kept on getting worse in the rising action, he can finally start resolving conflicts after the climax. *Page 13*

Conflict Resolution

Conflicts are either resolved or unresolved. When they are resolved, it is reasonably clear whether the protagonist or antagonist has won, i.e. in whose favor the conflict ended, as defined in the conflict's resolution-goal. This is still unclear when unresolved. *Page 33*

Dénouement

The fifth and last part of Freytag's pyramid. It shows the results of the outcome of the story's conflicts, e.g., 'and they lived happily ever after.' *Page 14*

Dramatic Phase

The part of Freytag's pyramid the story is currently in, according to the Virtual Storyteller. Since the climax is considered a point, not a phase, within the VST, only the other four parts of the pyramid have a corresponding phase. *Page 36*

Dramatic Structure

The parts in which a generic story can be divided, describing what happens in each part. *Page 7*

Emergent Narrative

A method in story generation in which the actions taken by the characters in the story are motivated by their goals, plans and emotions, with the story following from those actions, instead the characters' behavior following from what is necessary for the story. *Page 6*

Event

An out-of-character operator that describes something taking place in the world that is not the direct result of something a character did, e.g., a hurricane breaking out. *Page 20*

Expectation-schema

An schema that describes expectations characters might have over future changes in the world. *Page 21*

Exposition

The first part of Freytag's pyramid. In it, the main characters and the story's setting are introduced to the audience. It ends with an inciting moment, in which some event or action occurs that introduces the story's primary conflict. *Page 12*

Fabula

A graph maintained by the Virtual Storyteller's plot agent, that contains all operators that have been performed within a story as well as, if applicable, the motivation behind them. *Page 23*

Falling Action

The fourth part of Freytag's pyramid, in which all still running conflicts are resolved. *Page 14*

Framing Operator

An out-of-character operator that introduces new elements to the world, of which it is pretended that they have always been there. *Page 20*

Freytag's Pyramid

The name of the dramatic structure introduced by Gustav Freytag. In splits a story into five parts: Exposition, rising action, climax, falling action and dénouement. *Page 12*

Goal Manager

A part of the plot agent. Goals can be assigned to it, after which it will make a plan and perform it in order to achieve that goal. Used by the plot thread manager to start plot threads. *Page 29*

In-character Operator.

An operator that represents a change to the world that is directly performed by a character. *Page 19*

Inciting Moment

The event at the end of the exposition that either starts the story's primary conflict, or explains why it suddenly becomes relevant. *Page 12*

Inference Operator

An in-character operator that can be used in a plan to draw conclusions about the current state of the world. *Page 34*

Narrator

A system that can read the fabula generated by the Virtual Storyteller, and present it in some human-understandable form. *Page 17*

Operator

A structure that describes a possible change to the world. It consists mainly of preconditions, that describe what the world should look like before the operator is executed, and effects, that describe what the world looks like after execution, i.e. how the operator changes the world. The operators used in this thesis are actions, events, framing operators and inference operators. *Page 18*

Out-of-character Operator

An operator that represents a change to the world of which the cause can not be directly attributed to a character. *Page 20*

Perception

A change in the world that is noticed by a character, or that he believes he noticed. *Page 21*

Plan

A list of operators and the partial order in which they should be executed, the effects of which should result in the achieval of a goal. *Page 21*

Plot Agent

An agent that mediates between the character agents and the world agent. It passes on operators from the characters to the world agent, and their results from the world agent to the character agents. This thesis expands its functions making it also manage plot threads, and therefore the conflicts in a story. *Page 22*

Plot Thread

A structure that describes a coherent, human authored, partial story world state, and what can happen once the world is in that state. This structure is used to describe conflicts in. *Page 28*

Plot Thread Manager

A part of the plot agent that starts, manages, and keeps track of plot threads. Since conflicts are described in plot threads, the plot thread manager is directly responsible for managing the conflicts in a story. *Page 30*

Primary Conflict

The most important conflict of the story. *Page 12*

Resolution-goal

A goal that describes the conditions under which its corresponding conflict can be considered resolved. This is given as a goal to the characters in the conflict to ensure that a goal will be resolved. *Page 33*

Rising Action

The second part of Freytag's pyramid. In it, further obstacles for the protagonist in solving the primary conflict are introduced, in the form of secondary conflicts. *Page 13*

Round

A discrete unit of time in the Virtual Storyteller. Once every round, the plot agent requests the characters to send them an operator they want to perform. Some operators require more than one round to perform, making the character performing it not able to do any other in-character operator in the meantime. Other operators require no rounds at all to perform, allowing the character to perform multiple operators in a round. *Page 22*

Secondary Conflicts

Additional conflicts, besides the primary conflict, that form obstacles for the protagonist in resolving the primary conflict. *Page 13*

Virtual Storyteller

An emergent narrative project to which the research in this thesis is directly applied. Also abbreviated as VST. *Page 16*

VST

Abbreviation for Virtual Storyteller. *Page 16*

World Agent

An agent in the Virtual Storyteller that keeps track of the state of the story world. *Page 17*