Hide and sneak

Perceptions in the virtual storyteller

Hans ten Brinke

April 16, 2014

Graduation committee
dr. M. Theune
prof. dr. D.K.J. Heylen
J.M. Linssen, MSc

"Voor Henk-Jan. Ik wou dat je er ook bij was"

Abstract

The Virtual Storyteller is a multi-agent system that can generate stories. It does this by simulating the actions of autonomous characters in a virtual storyworld. The characters can select goals that have been provided by a storyworld author and will plan actions to achieve those goals. My work on the Virtual Storyteller aims to make the characters more believable by limiting their perceptions to what they should realistically be able to see.

This thesis describes a set of visibility rules that the character agent applies to incoming perceptions. The agent still keeps a complete and accurate version of the storyworld in memory which the agent can use on an actor level. The character however uses an incomplete and possibly incorrect version of the storyworld to make plans. To help the character make plans in case it does not have all the required knowledge available in-character, an assumption operator is introduced. This allows the character to make assumptions about the location of objects to prevent it from deciding that a goal is no longer achievable. Using these assumptions, the character can search for objects in the storyworld.

Whenever a character has to decide where to look for an object, there is an ideal opportunity for the actor to alter the character's course without sacrificing believability. A second implementation of the assumptions makes use of this opportunity by guiding the characters towards conflict in the first half of the story and towards a resolution in the second half. Using this technique ensures that a conflict will almost always be played out, regardless of the starting locations of characters and objects in the storyworld.

Preface

When I was looking for a subject for my Master Thesis, the Virtual Storyteller was one of the first projects to catch my attention. Since then I have done two research topics as well as this thesis on the subject and I am still not out of ideas. Making a computer write stories is an amazing experience in itself. But besides storytelling, I am also interested in games and especially in games with great stories that also give the player a lot of freedom. How cool would it be to play a game in which every decision you make has an impact on the story? You could play it many times without getting bored by the story.

Now the story of my studies at the University of Twente is finally coming to an end. I would like to take this opportunity to thank the people that have made my work possible. First of all, thanks go to my parents, who gave me the chance to study in the first place. I would also like to thank my girlfriend's parents who have been very supportive along the way and most importantly during the final stretch.

A lot of gratitude goes to my graduation comittee. First of all, Mariët Theune, thank you for all the support and the endless patience. Jeroen Linssen, thank you for filling the spot left by Ivo, your advice on my thesis has been very helpful. Thanks also to Dirk Heylen for guiding me in the right direction during the early stages of the project and for providing support when it was needed. To all members of the Storytelling group, past and present I would also like to say thanks, it has been an inspiring experience. Especially I want to thank Ivo Swartjes for his great work on the Virtual Storyteller and his useful advice and technical support.

Finally I would like to thank some people who have helped me to see my studies through to the end, Kees Ketting, Frank Droste and Fata Proost. Your support has been invaluable and without you I might not have made it this far.

Most important of all, I would like to thank my girlfriend, Ilse Slag, for sticking with me through good times and bad times. We've been through a lot together, but now we can close this chapter with a happy end and look forward to the next part of our story.

Hans ten Brinke
April 2014

Contents

1 Introduction

Stories exist in many forms, from oral narrative to written literature, classical theatre and modern movies, comic books and videogames. For as long as language has existed, man has been able to tell a wide variety of stories. Computers have become an increasingly popular medium for storytelling, be it as the main medium in for instance games and weblogs, as an alternative for traditional media as ebooks or as part of cross-media projects. But is it also possible for a computer to create interesting stories?

Several systems have been designed over the years that allow a computer to generate stories. One of these systems is under development at the University of Twente. It is called the Virtual Storyteller (VST) and it started out as a master project by Sander Faas in 2002 [Faa02]. The Virtual Storyteller is a multi-agent framework designed to generate and present stories. Its structure is inspired by an Improvisational Theatre form called Typewriter. In a Typewriter scene there are several actors, one of whom is assigned the role of narrator and director. The director is responsible for setting the overall structure of the story by providing a location and adding characters and the other actors make use of improvisation to fill in the plot.

The Virtual Storyteller uses an approach similar to Typewriter. A plot agent takes on the role of director and is responsible for the setting, assigning the roles and keeping track of the plot. One or more character agents try to fill in the plot by making plans to fulfill their goals or reacting to situations that arise. The original project was expanded upon by students and their advisors in smaller as well as larger projects. In 2006 the basic architecture of the Virtual Storyteller was redesigned to its current structure [Swa06, Swa10]. This thesis describes my contribution to the VST. In this chapter, the Virtual Storyteller will be briefly introduced. After that the goal of my work on the VST is explained and finally some initial research questions will be formulated.

1.1 The Virtual Storyteller

The VST generates stories through the simulation of a virtual storyworld with autonomous characters. This approach is called emergent narrative: the story emerges from the actions performed by the characters, motivated by their goals and emotions. Figure 1.1 shows a global overview of the Virtual Storyteller. The process of generating a story has been divided into three parts: simulation, discourse generation and presentation. In the simulation part the plot is written by simulating the actions of characters in the storyworld. The world agent keeps track of the actual state of the storyworld as well as all actions and events that are taking place at any given moment. The character agents can take on goals that they will then try to achieve. The VST addresses the issues of causality and coherence by logging the story not directly as text. Instead the plot agent records the plot as a structure of causally linked events (called a fabula in narrative theory), that forms the content of the story. The discourse generation part takes this fabula structure and expresses it in natural language, turning it into a proper story [Sla06]. Finally, the presentation part presents the story to the user. This can be done in different forms. The story can be presented as plain text but it can also be converted to speech and read to the user by an embodied storyteller agent. Work has also been done to express the stories in comic form [Zee10]. At the time this project was started, the VST lacked the possibility for user-interaction, but an interactive version has since been developed [Alo12]. It allows the users to take control over any of the characters. A map of the storyworld is projected onto a multitouch table and characters can be moved by drag-

ging them from one location to another. Available actions can be chosen from a list. My work however has been done on the non-interactive version. This thesis will focus on the simulation part of the storyteller, which is described in more detail in Chapter 2.

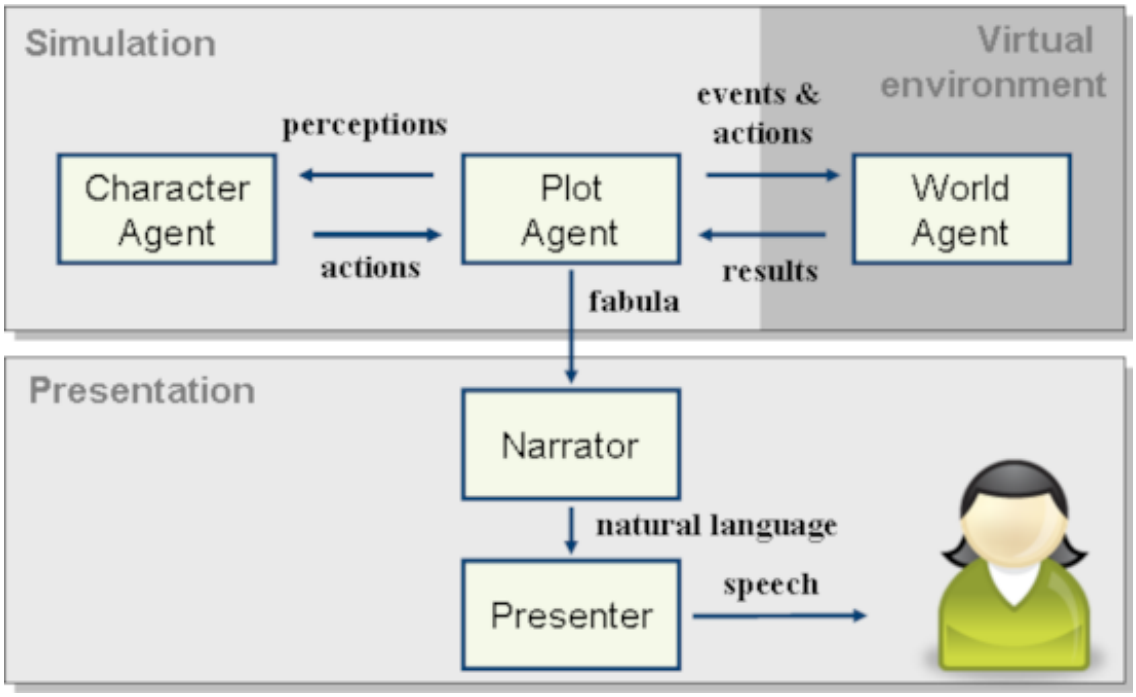

Figure 1.1: Global architecture of the Virtual Storyteller [Swa10]

1.2 The problem

On the surface, two stories may appear different. The characters, places and events in one story are different from those in the other. However, below the surface, stories share many common elements, patterns and structures. Aristotle describes a story as a complete causal chain of events. It has a beginning for which the cause is not important to the story, a middle for which the events have a cause within the story and which themselves are the cause for further events within the story and an end that causes no further events relevant to the story[Ari07]. Freytag describes a dramatic structure of five parts also known as Freytag's pyramid: exposition, rising action, climax, falling action, and dénouement [Fre00]. Campbell describes a pattern found in many stories from around the world, the monomyth, also referred to as the hero's journey [Cam49]. On a smaller scale, authors also often apply common techniques to move a story forward, to drive the characters to action. Although put into very different contexts, many situations in stories have very similar underlying structures. A lot of examples of such conventions, also referred to as tropes, can be found on TV Tropes [tvt].

The stories generated by the VST still have much room for improvement. Many types of elements that are commonly used in stories by human authors are currently beyond the VST's capabilities. One of the problems that is preventing many of these elements from being used is the fact that characters in the VST are omniscient. Almost all human authored stories, whether they are fairy-tales or complex literary works, rely on the fact that characters are not omniscient. A person is missing, an object stolen, a character tries to reach his goals by deception, there are many elements where false or missing information is key to a plot.

Take for instance the fairytale of the wolf and the seven little goats. Grandma goat leaves the seven little goats at home and forbids them from opening the door for anyone except her. The

wolf wants to take advantage of this and eventually manages to trick the little goats into letting him in by disguising his paws and his voice. Only one of the goats escapes by hiding inside a grandfather clock. Neither the wolf's disguise or the little goat's hiding spot would have worked if the characters were omniscient.

Vladimir Propp made an analysis of Russian Folk Tales by breaking them down into morphemes. He discovered 31 recurring functions that make up the narrative structure of these stories. Trickery is one of them, but many of the others also rely on deception or misinterpretation [Pro68]. They would not work if the characters knew everything. If a false hero would attempt to claim the hero's prize or marry the princess, no-one would believe his lies. A character would not fall into a trap if he knew about it. Stories in the detective genre are based on the fact that the detective does not know who committed the crime. The problem of omniscient characters has to be solved in order for a number of other concepts to work:

- Deception
- Character believability
- Emotions
- Social interactions

What a character has or has not perceived plays a key role in their behaviour. While writing one of the first story generation systems, TALE-SPIN [Mee77], Meehan already encountered the problem of perception. He overlooked the concept of noticing when writing the inference rules for his program and the following mis-spun tale resulted:

> Henry Ant was thirsty. He walked over to the river bank where his good friend Bill Bird was sitting. Henry slipped and fell in the river. He was unable to call for help. He drowned.

Bill bird was meant to rescue his friend, but Henry could not ask for help because the rules of the program prevented talking when in water. Bill did not notice his friend falling in the river, he would only respond if he was asked a direct question. The problem with the VST is the opposite, characters notice everything even when they should not. The main reason for this is the way agents perceive the storyworld. At the beginning of the simulation all characters have a full and accurate model of the storyworld in memory, and as actions are performed, the results of those actions are broadcast to all characters. Thus there is no way to keep information hidden from a character. If a character tried to deceive another, the other would immediately see through this. Apart from making deception impossible, omniscient characters are also less believable. If a character on the other end of the storyworld is moving the object the hero is after, the hero will instantly know this. Instead of going to where he should believe the object to be, he will alter his plans and go toward the object's new location.

Character emotions are currently not being simulated by the VST, but in future versions they are expected to play an important role in the characters' decision making process. Character omniscience will be a considerable problem there as well. Characters will form emotions based on perceptions and beliefs. Imagine a storyworld with several characters all doing different things at the same time. If they had to respond emotionally to every event in the world, instead of just those that they can see, it would become confusing to the audience.

Characters in stories often don't act alone. They will turn to others for information or advice. However if all characters have the same knowledge, there will not be much for them to tell

each other. Before we can think of implementing social interactions, we will need to create an environment in which they are useful. This will be required in order to implement concepts like trust and betrayal or the possibility of eavesdropping which often occur in stories.

1.3 Goal

The goal of this project is to allow the character's knowledge of the world to differ from the actual state of the world. The perception system is altered to allow the results of actions to be hidden from characters. It should be possible for a character to run into a situation that is different from what he believes to be true. At the same time, the characters have a role as actor for which they require complete and correct information about the world. To facilitate this we separate the agent's knowledge into actor knowledge and character knowledge. In the resulting system characters will have to find ways to discover the missing information. The ultimate goal is to improve the stories the VST generates. In order to evaluate whether the system can generate interesting stories with non-omniscient characters, a new story domain is created. So the following is needed:

- Perception system
- Separate knowledge bases for actor and character
- Reasoning skills to help characters find missing information
- Story domain for testing
- Evaluation

The project time has to be divided between these activities. Because of this, the reasoning abilities will only get limited attention and the resulting behaviour will not be very sophisticated. Social interactions could also be considered to allow characters to share their beliefs with each other, but this would require additional social reasoning skills. Researching and implementing those would likely be enough for a separate thesis so they will be outside the scope of this project. To compensate for this, the evaluation stories will use relatively simple characters that will not require complex behaviour. During the evaluation we will also try to deceive the characters. Deception can take many forms, some more complicated than others. Most of them require that the characters have insight into each other's behaviour to work effectively. This includes knowledge about what the intended victim knows, their decision-making process, plans and goals. Much of this is currently missing from the VST so we can't implement complex forms of deception yet. Therefore, the new perception system will be tested with relatively simple forms of deception such as hiding objects and giving false information about the location of objects.

One of the existing story domains, the pirate domain, will be adapted to form the new story-world. The original pirate characters are replaced with simple cartoonish characters. A rat has invaded the pirate ship and is stealing food supplies. A few pirate insults later the name Scurvy the Bilge Rat is born. In order to get rid of Scurvy, a cat named O'Malley is brought on board. The story will follow these two as they try to find objects or one another, or hide from the other using the perception system. The domain is partly inspired by cartoons like Tom and Jerry, mixed in with the existing pirate domain.

1.4 Research methodology

My goal for the Virtual Storyteller is to allow deception of characters by blocking perceptions and allowing for characters to have false beliefs. The main research question is how to make this possible without breaking the character's ability to plan towards reaching their goals. Research question 1 will be divided into two parts.

Research question 1a: How can perceptions be blocked for characters?

Research question 1b: How can the character be made to deal with missing information?

In order to answer these questions we first have to investigate how perceptions originate and are handled by the system. During the design of the plot agent it was argued that the plot agent should have ways to steer the direction of the plot, one of which was the ability to influence character perceptions. This resulted in the implementation of a perception manager which could eventually be improved to block perceptions from characters. However, since this original design, the VST has evolved. One of the first questions will be whether the plot agent is the right candidate to decide what characters can see.

Character agents have been given a dual role as actor and character and now have more responsibility for the plot. In order to support both roles, the agent will require two sets of knowledge: an accurate one for the actor and a more believable incomplete one for the character. The character agent's knowledge base will be divided into in-character knowledge and out-of-character knowledge. This results in a system where perceptions can be hidden from characters and deception becomes possible without depriving the actor of the accurate and complete knowledge it requires for out-of-character behaviour such as justifying goals and using framing operators. However, as expected another consequence is that characters can get stuck when they no longer know where an object is. In the old system this would mean the object doesn't exist. In the new system, the object may exist without the character knowing. The character will either have to find out where it is or fail to make a plan. A method has to be devised to allow the character to search for the missing information in a believable way. Methodical searching of the storyworld may give a solution to the planning problem but won't nescessarily improve the resulting story.

We have looked at other storytelling systems similar to the VST. Do they deal with issues of perception or deception at all? And if so, how do they handle planning when a character no longer has all the knowledge required to make a plan? The systems that were examined that deal with this problem all use a very different method of planning. Changing the planner entirely is not an option so a different solution has to be found. We will allow characters to make assumptions about the missing facts. These assumptions don't always have to lead straight to the intended target, otherwise the situation would be no better than before the new perception system. The assumptions can be used to give a character false information and steer the plot in more exciting directions by making use of the actor's knowledge. This way the planning problem can be solved and at the same time it gives an opportunity to steer the story towards a more interesting outcome.

1.5 Thesis outline

In Chapter 2 the inner workings of the VST are examined. This provides better insight into where and how to block perceptions and what the consequences are. Chapter 3 gives a description of some other approaches to interactive storytelling and how perception and deception play a role in those systems, and whether their methods are applicable to the VST. Opponents in computer games also need methods of limiting perception so Artificial intelligence for games is also examined briefly. Especially the role of an agent as a participant or as an actor is examined. Chapter 4 describes the changes that have been made to the Virtual Storyteller to make the new perception system work. In Chapter 5 a new operator is introduced that allows a character to make assumptions about the missing facts. In Chapter 6 the approach is applied to Scurvy's domain to demonstrate the authoring and to evaluate the results. Chapter 7 presents conclusions and some ideas for future work.

2 Perceptions in the Virtual Storyteller

This chapter will give a more detailed look at the Virtual Storyteller and how it generates stories, giving special attention to the role of perceptions in the system. The second half of this chapter discusses how changing the perception system will affect the rest of the system, and where changes need to be made.

2.1 Overview

As mentioned in Chapter 1, the Virtual Storyteller is a multi-agent framework that generates stories by simulation. The plot generation is split up between three parties, as shown in Figure 1.1. The character agents represent the characters of the story. They are given goals to achieve in order to advance the plot and they will try to make plans to reach these goals. The world agent is responsible for keeping track of the world as it is as well as any operators that change it. The plot agent was originally intended to take the role of the director from the typewriter scene. In this capacity it has the ability to influence the direction of the plot by rejecting requests from the characters or blocking perceptions. In practice, these options are not being used as all actions are approved and perceptions are always passed on to every character. Currently the most important task of the plot agent is to record the plot, building the fabula layer that the narrator can transform into a story [Swa06]. A more detailed explanation of each of the agents follows below.

The agentlauncher is the central interface that can be used to select the story domain and launch the JADE (Java Agent DEvelopment) [jad] agent platform on which the agents run. After selecting a story domain and starting a plot and world agent, the plot agent takes control. Character agents can be started manually, or will be started by the plot agent when required. The story generation takes place in discrete turns, initiated by clicking the next turn button on the plot agent or activating an automated loop. The knowledge base of the agents makes use of the SWI Prolog Semantic Web Library [Wie], using JPL to connect the Java layer to the Prolog layer.

2.2 The storyworld

The world agent is responsible for keeping track of the environment in which the story takes place. Each agent has a knowledge base which includes a personal copy of the storyworld. However, the world agent has the central copy and is the only one that can directly execute operators on it. The other agents can only request operators and wait for the results.

2.2.1 The Ontology

The virtual world is represented by a list of facts that represent objects and the relations between them. The possible objects and relations that can be used to create a specific storyworld are defined in an ontology. This ontology is written in the OWL web ontology language [owl], an extension to the resource description framework, RDF. To improve the reusability of the ontologies, a distinction is made between core facts and domain specific facts. The storyworld core ontology defines core concepts at an abstract level, e.g. objects, pathways, containers. It also contains information that defines how objects relate to each other. The fabula ontology describes how the fabula is constructed out of basic elements like goals and actions; more about the fabula

further on in this chapter. It also contains a basic description of the different action types, though the actual implementation of the actions is done in prolog, using schemas as explained below. Figure 2.1 shows part of the ontology for the Scurvy domain, including the core and fabula ontologies it builds on. Using the facts from these ontologies, a storyworld instance is created using "Turtle" or Terse RDF Triple language. This instance represents the state of the storyworld at any time during the story. It is built up of RDF triples of the form: (subject, predicate, object). Some examples of RDF-triples:

```
( scs:scurvy rdf:type scs:rat )\\
( scs:cheese01 rdf:type scs:food )\\
( scs:scurvy swc:has scs:cheese01 )\\
```

This signifies that Scurvy is a rat, the object `cheese01` is a type of food and Scurvy is holding `cheese01`. In this example `cheese01` is an identifier, but it doesn't have to be an actual piece of cheese since the ontology does not go into enough detail to list cheese as a subclass of food. Note the different prefixes used in subject, predicate and object. The prefix determines which ontology the object is from: scs for the scurvy setting, rdf for the resource description format, swc for storyworld core.

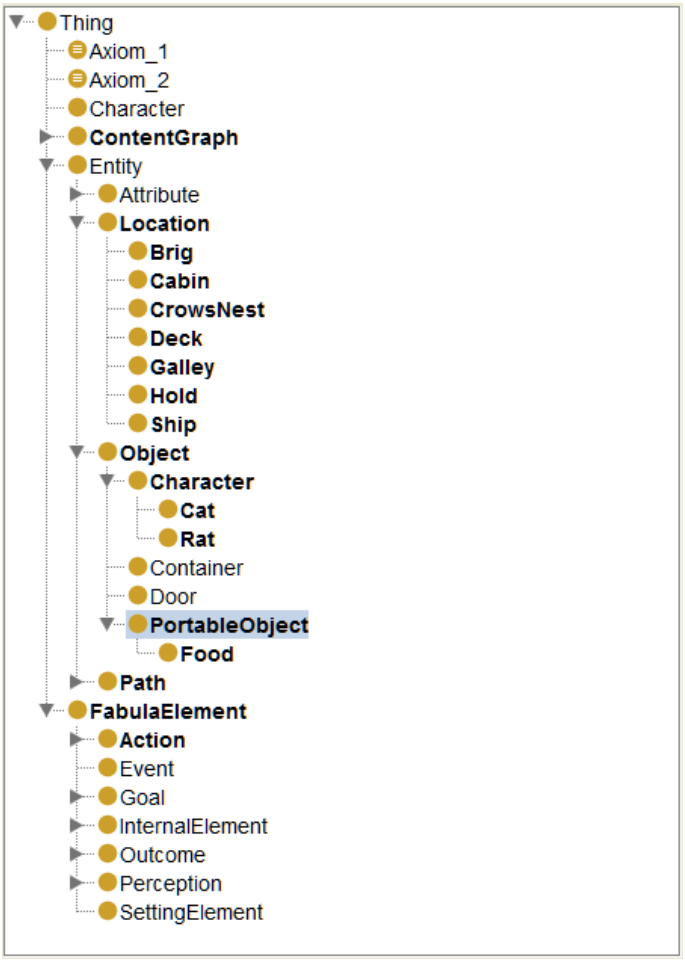

Figure 2.1: Part of a domain specific ontology

2.2.2 Operators

In order to reach their goals, characters have to be able to change the world. The actions and events are defined as STRIPS-like operators [FN72]. Every action has preconditions, effects and a duration. Preconditions are represented as a list of conditions. Each condition has a truth value and a list of subconditions. The truth value signifies whether the subconditions must be true or false in order for the condition to hold. A subcondition may query a fact or fabula triple, or apply a rule. Rules are written in prolog to define more complex relations. For example, the typeorsubtype rule is an OWL rule that can be used to determine if an object is of a certain class or a sublass thereof. The preconditions of a dress action use this rule to determine whether the object to be worn is a subtype of the `wearableproduct` type. The `wearableproduct` type is defined in the core ontology and subtypes can then be defined in a storyworld setting ontology, which contains domain-specific knowledge. A fairytale setting may define a crown as a `wearableproduct`, in a pirate setting it could be an eyepatch.

Dress	
Preconditions	
true	rule(Agens, knowsAction, Dress) rule(Patiens, typeOrSubType, WearableProduct) fact(Agens, has, Patiens)
Effects	
true	fact(Agens, isWearing, Patiens)
false	fact(Agens, has, Patiens)

The effects of an operator specify lists of triples that become true or false upon sucessful execution of the operator. The world agent keeps track of all operators that have been scheduled for execution. Upon starting an operator the preconditions must be fulfilled or the operator is cancelled. For operators with a duration, i.e. actions and events, the preconditions must hold for the duration or the operator will fail. For instance when a character wants to walk from one area to the next, a precondition is that the path must not be blocked. If for some reason the path becomes blocked before the duration is over, the action is canceled before any effects are applied. If not, the effects are applied and the character is removed from the starting area and added to the destination area. Events are similar to actions, they can make physical changes to the world and they have a duration. Unlike actions however, events are not directly performed by characters although they may result from character actions and they may be used in plans. Take for instance a pirate character Billy Bones who has a goal that his enemy's ship is destroyed. He has no actions to perform that directly destroy the ship. He can however plan an action to fire a cannon at the ship, followed by an event in which the ship sinks, causing it to be destroyed. The event has a precondition that a cannon has been fired at the ship. Billy cannot execute the event himself, he will have to request the plot agent to perform it. The latter can then decide whether to accept or deny the request, although as mentioned earlier, in practice the plot agent always accepts these requests [Swa06].

Sink	
Preconditions	
true	rule(Agens, typeOrSubType, Ship) rule(Instrument, typeOrSubType, Cannon) fact(Instrument, shotAt, Agens)
Effects	
true	fact(Agens, sunkBy, Instrument)

Apart from actions and events, there are other means by which to change the world. The initial setting of the storyworld has a big impact on the diversity of stories that can be simulated. To

reduce this impact, the VST makes use of a technique called late commitment [ST09]. Instead of completely defining the entire setting at the start of the story, parts of it are filled in as needed when the story progresses. For this purpose, framing operators were introduced. They have a very similar structure to actions and events, however they are interpreted differently. The effects of a framing operator establish a fact as though it had been that way since the story started. Therefore a framing operator has no duration and characters do not react to the effects. In the eyes of the character nothing has happened, the world was always like this. Characters can use framing operators for two purposes. If a character has no goal and no possible goals left to choose from, he can justify a goal by using a framing operator to fulfill an open precondition of that goal. For instance, if a goal specifies a character needs to be the captain of a ship to adopt that goal and there is no captain specified for the ship yet in the story, a framing operator can be used to make someone captain, after which they can adopt the goal. If a character can no longer find a plan to reach his current goal he can use a framing operator to change the setting so that a plan becomes possible. For example if a character wants to kill another character but has no weapon, they can use a framing operator to add a weapon to the storyworld. Special care has to be taken when authoring these framing operators to prevent introducing inconsistencies with the story so far.

2.2.3 Perceptions

The world agent is responsible for keeping track of all operators that have been scheduled for execution. Every turn, the world agent checks the list of scheduled operators for any that have completed their duration. If the preconditions still hold, the action was succesful and the effects are applied, otherwise the action failed. This is where character perceptions originate. The results are sent to the plot agent which will process them and turn them into perceptions to be distributed to the characters.

When the plot agent receives an operator result from the world agent, it is sent to the perception manager, which checks whether the action was finished and if so registers the operator and effects in its operator history. This history is processed each turn to determine for each character which operators that character still has to be informed of. Whenever it finds such an operator, the effects are converted into RDF-triples, which are then turned into perception objects and sent to the character. The awareness map is updated to reflect that the perception has been sent to that character so that it is not sent again later. No checks are made whether the character would logically be able to see the action or event.

Once a character receives a perception it is sent to the interpretation module. This module processes incoming perceptions and setting changes. The RDF-triples contained in the perception are stored in the character's personal copy of the storyworld. Fabula-elements for the perception and the beliefs that follow from it are added to the character's episodic memory. A reinterpretation step follows, during which a character might derive additional beliefs from the initial perception. For instance, seeing a character who is lying on the deck might be a reason to interpret that character is dead. However no rules for reinterpretation have been implemented yet so for now only the initial beliefs are returned. In fact, agents don't need to reinterpret any beliefs as they already know everything about the storyworld through the perceptions they receive. If there was an action or event that caused the character to die, everyone would have received a perception of this. This will change once perceptions can be blocked.

2.3 The characters

The character agents are semi-autonomous entities. They follow the belief-desire-intention software model. The agent has beliefs which are the facts the agent assumes to be true about the world. Desires are general goals the agent has. Intentions are those goals the agent has committed to achieve. Initially character agents were only responsible for believable behaviour. Their actions would only have to be consistent with their personality. The plot agent was given the

task of ensuring a well-structured plot. More recent work however has pointed out that coupling autonomous characters with the guidance of a drama manager is problematic [ST09]. Therefore, some of the responsibility for the plot has been shifted to the character agents. This corresponds to the ideas of improvisational theatre, where the characters share responsibility for the plot but none of them can directly control the outcome of the story. The character agent is given two roles: the character in the story (in-character, IC) as well as the actor that plays this character (out-of-character, OOC). The framing operators mentioned earlier are an example of out-of-character behaviour. Figure 2.2 shows a global view of the character agent. The character process as shown in the image has actually been extended to an actor process, but the basic principle remains the same. The interpretation module was introduced in the previous section. After it has processed all perceptions for that turn, the actor process selects what action to perform. Reactive behaviour takes precedence over planned behaviour. The reactive process handles immediate reactions to the character's surroundings by triggering rules, e.g. when someone greets you, you greet back. The deliberative process is tasked with handling planned behaviour. It appraises the current situation and selects which goal to act upon. Then it copes with the situation by making a plan to achieve the goal and selecting an executable action from that plan.

2.3.1 The planner

The deliberative process makes use of a partial order planner to choose actions that will achieve the character's goal. It creates a basic plan consisting of a start step which is the current situation, and a final step which is the desired situation. It then recursively chooses either operators or previous plan steps that fulfill open preconditions in the plan, or the start step if it already fulfills the preconditions. Causal links are added between each step and the precondition it fulfills. Ordering constraints ensure the correct order of execution and threat resolution ensures that the new step does not break any causal links by removing effects that are needed further on in the plan. This is repeated until all preconditions are fulfilled. The planner can insert IC operators (normal character actions) as well as OOC operators (events the character did not intend to happen such as a storm that blows his ship off course, or framing operators) into a plan. When choosing an operator to insert, existing plan steps and new IC operators are considered before OOC operators. The planner will explore all possible plans consisting of a given maximum number of steps, starting at 1, and increase the maximum only if no plan can be found.

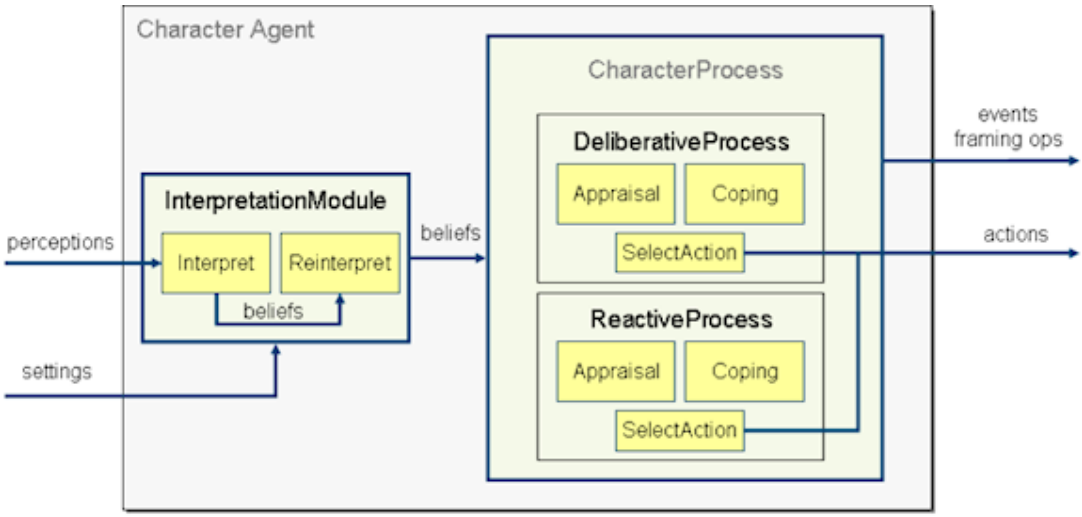

Figure 2.2: Architecture of the character agent [Aut]

In order to prevent far-fetched behaviour, OOC operators have to be treated differently from IC

operators. A character may not perform IC operators solely to fulfill the preconditions of an OOC operator. The character should have no knowledge of OOC behaviour and should not be trying to assist the actor unless it does so with an IC motivation. For example, there are two characters who hate each other. One of them decides he wants to kill the other, but he has no weapon. There is a framing operator which says a rapier can be added to the world, but only for a pirate. So the character signs up to become a pirate, then frames a rapier. Once he frames the rapier, it is treated as if he always had it. So there was no apparent motivation for him to become a pirate. If there already was another IC reason for him to sign up to be a pirate, it would be fine. In that case, the action will be added as a plan step for that reason and can then be used. If there is no such reason then the only other option would be to use a framing operator that said the character had also been a pirate all along. Therefore, when fulfilling an open precondition of an OOC operator, only other OOC operators or existing plan steps may be used. A second problem is that framing operators may introduce inconsistencies. For example, the hero is trying to get past a guard, and is not allowed to have a sword, so as not to arouse suspicion. Further on in the plan, the hero needs a sword after all, but since there is none, he frames one. Now a contradiction arises, because if he frames the sword, he always had it, so he would also have had it when he was walking past the guard. The effects of the framing operator contradict the preconditions of a planned action. When this happens, the framing operator should always be executed first. Further steps can then be planned to lift the contradiction. To ensure this, a causal link is added between the start step and each effect of a framing operator. The POP threat resolution will then order the steps correctly. In the example, the hero would for instance frame the sword at the start of the plan, then hide it to get past the guard and retrieve it later.

2.3.2 Goal selection

Aside from planning to reach the current goal, the character also needs to select the next goal from its desires. A goal selection module is presented in [TROdAH04] but this is currently outdated. During my work on the VST, a new goal selection model was created [Bra10]. Since it was completed well after my work had started, not much has been done to incorporate it into the additions I made to the system. The new goal selection module makes use of character traits and interests that provide positive or negative motivation towards selecting a goal. Goals are defined in schemas similar to actions. There are preconditions to adopt the goal and success and failure conditions that determine when the goal has been achieved or can no longer be achieved. These schemas can now be annotated with lists of positive and negative motivations which influence the choice a character makes when presented with multiple possible goals. A dramatic choice element is introduced that links the character's choice to its motivations in the fabula, so that the characters' motivations can be communicated to the audience. When one goal associated with a dramatic choice is completed, the alternatives are dropped. For example, a pirate character is ordered by his captain to dump the treasure overboard because the ship is being chased. The dramatic choice has two goals for the pirate to choose from: to dump the treasure as ordered, or to defy the orders. The first choice has positive motivations for loyalty and self preservation but negative motivations for greed, the second has the exact opposite motivations. The pirate has personality traits that assign a strength to each of these motivations, and by subtracting the value of the negative motivations from the value of the positive motivations, the importance of each choice for the pirate is calculated and the most important is chosen [Bra10].

Work has also been done to give more focus to coherent goal selection [Swa10]. To make the plots more coherent, whenever a character selects a goal it has to be causally related to the currently coherent plot. Something has to have happened in the coherent story to make the character choose this goal. Switching between goals also has to be believable, there has to be a cause for a character to suddenly change priorities. Imagine a character has a goal to cook dinner and a goal to keep her house tidy. She starts the cooking goal and takes a pot from the cupboard. Then she switches to the cleaning goal and puts the pot back in the cupboard. This is silly, of course she

should complete the cooking goal before switching. If on the other hand she was taking a pack of flour from the cupboard and accidentally dropped it on the floor, this event could trigger a goal to clean up the mess she made.

When a character no longer has any goals to adopt, because the preconditions of those goals aren't met, a goal can be justified. To justify a goal, the planner is used as well. However, since justification of a goal is an OOC process, the planner can only use OOC operators when justifying a goal. Executing the resulting plan will change the story setting so that the goal can be adopted. For instance if a goal to find a treasure exists, but no treasure has been placed yet, a framing operator could be used to put a treasure on a remote island [ST09].

2.4 The plot agent

In the first version of the VST, there was no plot agent and no world agent. The director agent was responsible for directing the plot, recording the events of the story so far and keeping track of the world. In [Swa06] this functionality was divided between the plot agent and the world agent. It was argued that the plot agent should have ways of understanding the emerging story and controlling its direction. As mentioned earlier in this chapter, some of this responsibility has shifted to the character agents, but the plot agent still plays a central role. Figure 2.3 shows a global view of the plot agent.

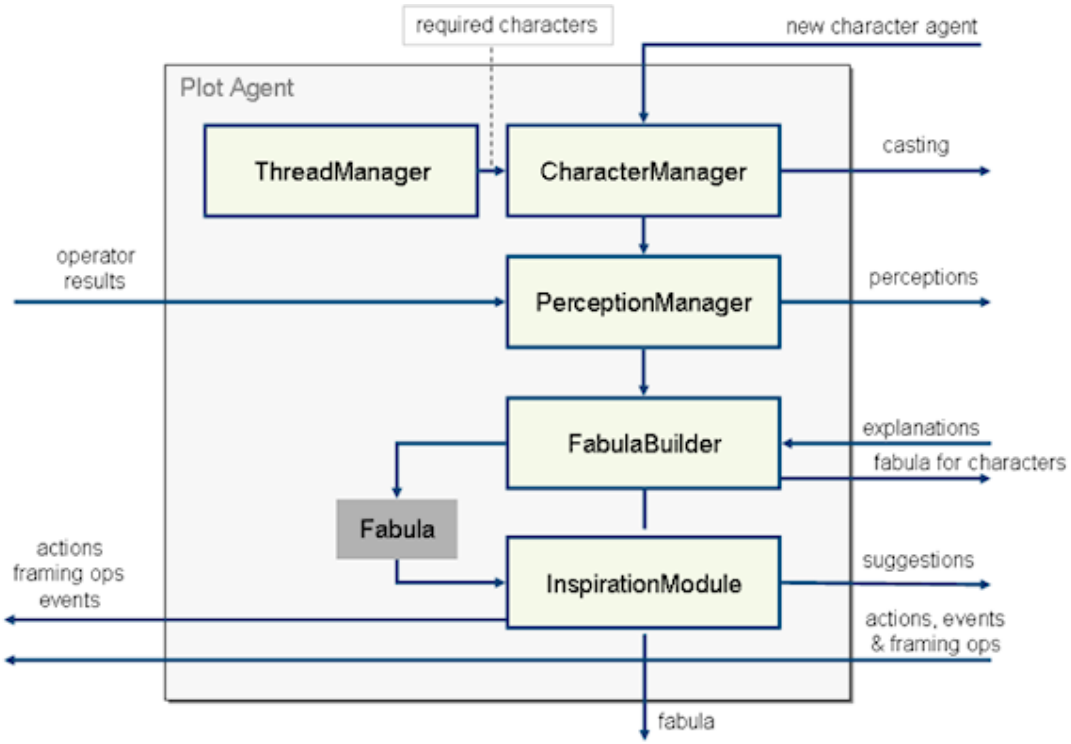

Figure 2.3: Global architecture of the plot agent [Swa10]

The plot agent sets up the premise of the story by starting up the conflicts that are essential to a story. These conflicts are defined in plot threads. A plot thread definition contains information about the setting, the characters and their goals in the thread. When starting up a thread, the thread manager determines what roles there are and informs the character manager. The latter casts character agents into the required roles, starting up new characters if needed. The world agent as well as the characters are informed of any setting elements defined in the thread. The inspiration module will suggest the goals from the thread to the relevant characters. The simplest

stories consist of just one thread, but recent work has allowed the introduction of secondary conflicts [Tom09].

2.4.1 Fabula

One of the main tasks of the plot agent is to record the plot. It uses a fabula model to do this. It records a network of causal relations between seven types of elements. Setting elements are aspects of the storyworld. Actions and events cause changes in the storyworld. Goals are what drives characters to these actions, and outcomes relate to the success or failure to achieve these goals. Perceptions are what a character sees of the world. Internal elements are what goes on inside a character, such as beliefs and emotions. These elements are connected by four types of causal relations. Physical and psychological causality represent causes that are unintentional, whereas motivation is an intentional cause. Enablement is a relation in which one element makes another element possible, the effects of the cause overlap with the preconditions for the result. Figure 2.4 shows the possible connections between each of the element types.

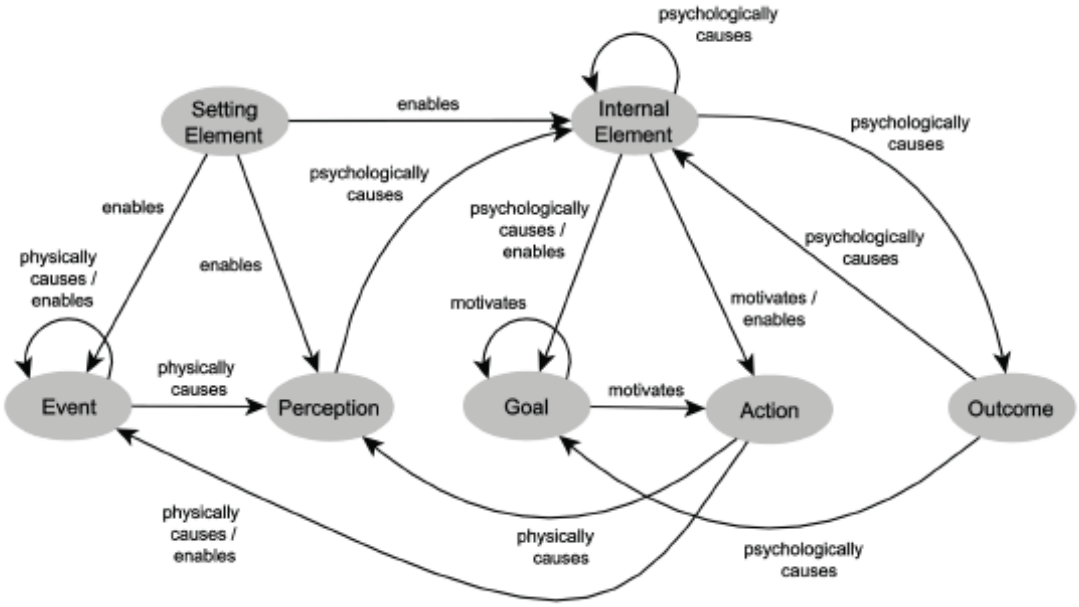

Figure 2.4: Fabula model [ST08]

Pieces of fabula are built as characters receive perceptions, form beliefs, take on goals and execute actions. These pieces are sent to the plot agent, where the fabula builder assembles them into its fabula graph. The narrator can use this graph to determine what elements of the fabula are relevant to the story and turn them into text [Swa06, ST08, Swa10].

2.5 Hiding perceptions

Perceptions form the basis for the characters' beliefs. Changing them will have consequences throughout the rest of the system. This section will present an initial idea of how perceptions can be hidden and what consequences this will have.

2.5.1 Senses

If we want to hide perceptions from characters, the first question that rises is how to decide what a character can perceive. Perception is the process of consciously registering a sensory stimulus. The traditional five senses are sight, hearing, smell, touch and taste, although there are others depending on the definition of what constitutes a sense. Animals sometimes have senses that are unlike any human sense, for instance the ability in some birds to detect the Earth's magnetic field to help them navigate. Beyond that, characters in stories sometimes have their own amazing powers of perception. For instance, Spiderman has his "spidey sense" which warns him of impending danger. Another example is clairvoyance, which is sometimes used as a plot device to give a character information they could not have gotten otherwise. Currently the perceptions in the VST do not trigger a particular sense. They contain facts about values of properties that are changed by an operator. As they are all passed on to the characters, there is no need to distinguish between senses. If we want to block a perception for a character, it will be because none of the senses could have believably registered the fact for that character. If multiple senses are implemented, then each property will have to be linked to one or more senses which it may trigger. This will be required because each sense will have a different method to determine whether it has been triggered. Where a visible change in the world may be perceived a lot later than the action that caused it, a sound made by that action is transient and can only be perceived at the time it is made. Some properties may not be linked to any sense, which means that the property can not be directly perceived and must be deduced from other facts. For instance, when emotions are introduced, a character may not always show his (true) emotions externally. A character should not be able to perceive another character's internal emotional state directly, unless we have a character with empathic powers (the ability to sense another's emotions), a plot device sometimes used in science fiction or fantasy.

The main reasons for implementing multiple senses would be to increase believability and to increase the amount of information a character can use. For this project, the main goal of blocking perceptions is to be able to hide information from characters. For this reason, we will start by implementing only the sense of sight. Expandability is a key feature of the VST, so an approach will be used that can easily be expanded with other senses. As storyworlds become larger and more complex, other senses or sources of information will get a more important role. Characters will need more ways to fill in the things they don't know. Hearing will become more important when social interactions are implemented. The Interactive Storyteller already has sounds although they were added mostly for entertaining the audience; nevertheless it may be interesting if the characters can hear them and respond appropriately. As shown by the examples, the traditional five senses may sometimes not be enough for a story, which is another good argument for an expandable approach. New and possibly strange senses can be added when a story needs them.

2.5.2 Visibility

As mentioned, we will focus on the sense of sight first. How should we decide what is visible for a character? The simplest approach would be to say characters can only see what is at their location, but that will not be enough. Some locations will logically have a view of nearby locations. For instance, a pirate up in the crow's nest should be able to see what happens below on the deck. A relation could be added to the ontology to mark which area has a view of which. However, this will not work if the view is conditional. For instance a character can see into the next room, but only if the door is open. This can only be checked if the visibility property were linked with the pathway that contains the door. In that case an additional property is not even needed, as the pathway itself can be used as a visibility relation, which makes it easier to include the state of the door in the condition. Additional visibility relations could be used in case the view does not correspond to a pathway but for instance a window, although a window could also be consid-

ered a path in some domains. In a modern storyworld, a security camera could be used as a link to provide a view of a remote location or possibly even a past event. This means that even on the level of a single sense, the system should be expandable, allowing multiple, possibly domain specific methods by which a character could receive a perception. Prolog rules can be used to describe the multiple methods by which a character can receive a perception. New clauses can be added to a rule when new methods of perception are added to the domain.

Aside from determining perceptions for a character, perception rules could also be used in preconditions. Take for instance an action to shoot another character. In the old situation, to prevent characters from shooting each other from the other side of the storyworld, you would have to add a precondition that the characters are at the same location. Using perception rules, this precondition could be replaced by a rule that determines you can only shoot what you can see.[1] Different rules could also be set up for the other senses. That way the smell of food could be used as a precondition for a "become hungry" event, which would allow a character to justify a goal to eat something. Or the rule could be used straight away as a precondition for the goal: "If I can smell food, I want to eat it."

2.5.3 Obviousness

In the previous section, we have only looked at the necessary conditions for having a perception. This prevents a character from seeing anything the character could not possibly have seen. We may also want to be able to hide perceptions a character could have seen. Sometimes, a character might not notice something that is happening right under their nose. For instance, a thief might be picking the hero's pockets. If the thief is always instantly discovered, this will not add much excitement to the storyline. This means it should be possible for characters to not notice something that they could have seen. Some events however are nearly impossible not to notice. For instance it would be unlikely for a pirate on deck to miss the fact that the ship is sinking. This suggests that it should be possible to determine the obviousness of a perception. Part of the solution could be to add an obviousness attribute to action and event schemas. However some effects of an action might be more obvious than others. For sound effects there may be a loudness to determine how likely it is for someone to hear it, for sight, the size of an object would be important. Other factors also play a role, such as what the target of the action is. Is the character who would receive the perception personally involved in the action? And what is he focused on himself? If the character who had his pocket picked tries to pay for something in a shop, he does not realise that his wallet is gone. His action will fail, but the world agent does not explain why it failed. If nothing changes the obviousness of the fact that the wallet is missing there is no new trigger for the related perception. The character would get stuck trying to get his wallet without ever knowing why he can't.

The second problem is how to make the decision whether or not to miss a perception. A random choice would increase the number of possible stories that can be generated, but it would make it difficult to reproduce the outcome of a particular story, making iterative authoring of a storyworld a lot more difficult.

Another idea would be to base the decision on what the result of the perception would be for the story by determining what the emotional result would be for the character. Emotions are an important part of stories, and while the characters in the VST have no emotional system yet, in the future such a system might be used to help guide the story as well as improve the believability of the characters. Such guidance could be accomplished by making a pre-appraisal of the results of a perception before deciding what to do with it. By comparing the emotional result of the perception to how we want the character to feel at that moment in the story, the decision could

[1] This would cause strange behaviour in a world where cameras allow visibility of remote places. If that were the case, a distinction would have to be made between direct and indirect visibility.

be made for the character to overlook something. For instance it could guide the hero character to be more open to perceptions that would cause negative emotions if the story is working up to a climax. Instead of adding randomness, this would actually contribute to the story.

At a more direct level, the actor could be given more influence over the character's behaviour. The actor might be given some goals that describe those things the author would like to see in the story aside from the goals that the characters are given. If a perception were to threaten such a goal, that would be a reason to let the character miss it. Implementing actor goals on top of the new perception system would be too much work for this project. Without a reasonable method to make the decision, the obviousness attribute would lose its purpose. Because of this, the obviousness attribute was left for future work so that more time could be spent on the planning problem described further on. Characters will see everything they could possibly see in this implementation. However, the idea of allowing the actor to guide the story towards an interesting outcome will be explored more in the next chapter.

2.5.4 Interpretation

As shown in paragraph 2.2.3, perceptions only happen at the time an action has been completed. This will be a problem when characters no longer receive all perceptions immediately. If a character enters a place where something has changed while he was not around to witness it, the character will not know about the change, nor will he be informed of it. The character may then repeatedly attempt an action which is possible according to his beliefs while in the actual world it isn't, resulting in an endless loop of failed actions. Characters will need a way to detect these changes.

Previously characters would also see all effects of an operator. For instance, a walk action has effects that the character is at the new location and no longer at the old location. If O'Malley believes Scurvy is on the Deck, yet now he sees him in the cargo hold, the belief that Scurvy is on the deck will have to be dropped. If only part of the effects of an operator are visible, the other effects should also be deduced. In this case, Scurvy can not be in two places at once, so even though O'Malley can not see the deck, he should still deduce that Scurvy is no longer there because he can't be in two places at once.

If we replace Scurvy in the above example with a generic object, O'Malley might assume the object he sees is a different object than the one he saw before and the original is still where he thought it was. This would be difficult to implement because all objects in the VST have unique identifiers. Making O'Malley assume it is a different object with an identifier that does not exist will be tricky, and from a storytelling perspective, the use of this scenario is dubious. The other way around, making O'Malley think it is the original object when it is not, is a more useful scenario as it can have an impact on the story when for instance the object turns out to be fake. This could be achieved by using inference operators.

For example, in the Redcap domain Grandma can make a plan to poison Wolf by giving him a poisoned pie. If Wolf knows about an object `pie01` which Red took to Grandma and he is given an object `pie02`, he might infer that it is `pie01` he was given and thus Grandma no longer has it. The difficulty is then how the character will discover that the object is fake. If he would try to use an eat action with `pie01` as the patiens, the world agent will reject the action since he does not have `pie01`. A solution might be to give the world agent or plot agent the ability to swap an action for a similar one with the same intent but a different outcome, for instance an `eatpie` action and an `eatpoisonedpie` action. Such actions might be grouped in a similar way as the goals in a dramatic choice are. So when Wolf attempts to use `eatpie` with `pie01`, the `eatpoisonedpie` action from the same group is substituted using `pie02` because the preconditions for that action state that the character should have a poisoned pie. Upon completion, the substitution will have to be reversed for Wolf's perception so that he will believe he no longer has `pie01` (which he

never had). This problem will not be solved in this project. In the current implementation, when an object is visible for a character, all triples concerning that object will also be visible, including the ones that are no longer true. For now we will ignore the fact that not all properties are visual as the other senses will not yet be implemented. All properties will use the rules for visibility.

2.5.5 Planning

The planner currently makes use of the fact that characters are omniscient. They know everything there is to know about the world. Because of this, if there is a possible plan in the current world state, it will be found. If the planner no longer has all the facts at its disposal when it starts planning, it may not be able to find a complete plan, which will result in a failed goal. This is different from the way a real person would plan. For instance, Jack is hungry. He decides he wants something to eat. When he leaves home to go to the supermarket he may not even know exactly what he is going to eat, let alone where exactly he is going to find it. He's not going to simply ignore his hunger and abandon his goal to eat. He just starts executing his plan to go to the store and fills in details about the rest of the plan later. He may find a nice product that is on offer and think of other ingredients to add. He will then start looking for those, or ask an employee where to find them. This does not mean that the characters in the VST should be programmed to behave exactly the same way, but a way will have to be found for characters to behave believably despite the fact that they don't have all the information. Chapter 3 will give an overview of a number of other storytelling systems and investigate how they cope with or avoid this problem.

2.6 Conclusion

In this chapter we have explained the importance of perceptions in the VST. They are a key element in the chain between performing actions and updating beliefs to reflect the results of those actions. Because of that they also play a central role in the making and executing of plans. If perceptions are hidden from characters, this will directly affect the beliefs characters have. Indirectly, it will have an effect on what emotions a character will receive when emotions are implemented. Furthermore, having false beliefs or missing beliefs will influence what plans a character can and will make.

Instead of implementing the traditional five senses, the choice was made to use an expandable approach, using only the sense of sight initially. This reduces the amount of work that has to be done to create a working perception system while at the same time allowing for more creative freedom to add some of the more exotic senses that are sometimes used in stories. A system of rules will be used to determine what is visible for a character and we ignore the fact that not every perception should be equally obvious. Whenever an object becomes visible for a character, all properties of that object will also become visible. All properties are also treated as visual regardless of which sense a property would normally trigger. Mistaking an object for another, similar object will not be possible. Chapter 3 presents an overview of several other storytelling systems as well as some insights from AI for computer games. In Chapter 4 the new implementation will be discussed in more detail. How we will deal with the planning problem is explained in Chapter 5.

3

Perception and deception in other storytelling systems

In this chapter we will describe a number of other storytelling systems, making note of how they handle perceptions and planning. This provides some helpful ideas on how to deal with the problem of planning with incomplete information. This is not a complete overview of automated story generation systems because for many of them there is no clear description of how perceptions are handled. As a related field of research where similar problems may be handled, game AI literature provides some insights into the problem as well.

3.1 The Oz Project

The Oz Project at Carnegie Mellon developed technologies for building believable agents for interactive storytelling. It has been discontinued as of December 2002, however former members of the project still work on similar projects. In [BLR92] the Tok agent architecture is summarized and a particular agent, Lyotard, is described. Tok is an agent architecture developed for use in an Oz world. Lyotard was designed to pass as a believable house cat in an Oz simulation, and displays goal-directed behaviour, reactivity, emotion and social behaviour. The Tok agent is divided into several subsystems, most notably Hap, Em and the sensory routines and integrated sense model. Hap handles reactive and goal-directed behaviour, while Em concerns itself with the character's emotions and social relations. The sensory routines collect sensory information which is then merged into the integrated sense model (ISM), which keeps track of the agent's knowledge of the current state of the world. This world is implemented as an object oriented simulation in which agents can perform actions by invoking methods on appropriate sets of objects. Objects are connected to each other by topological relations.

Tok agents run a three-step loop: sense, think, act. First the sensory routines obtain raw sense data from the world. Sense data objects are propagated from the item sensed through the world to the agent. These data convey properties of objects, relations between objects, events or actions. Sense data objects consist of a number of property/value pairs. Unique identifiers are not used, agents have to derive identity of the object sensed from the property values. Sense data can be transformed during transit to simulate, for example, muffled voices behind a door. Sense data comes in multiple modalities: sight, sound, smell, touch. The collected data forms a graph of the newly encountered world fragment, which is stored with a timestamp, while also keeping older graphs. Once the data has been collected, an attempt is made to merge these data into the ISM. This includes merging data from different modalities if they appear to be related, and merging old and new perceptions that appear to refer to the same object. If inconsistencies are found, the model is updated. For instance if sight suggests an object is soft and later touch reveals that it is in fact hard, the ISM is corrected. Data from the ISM as well as the continuously updated sensory information is queried while choosing actions or updating the emotional state of the agent.

During the think-step, Hap selects the agent's next action, based on perceptions, current goals, emotional state, behavioral features and other internal elements. Goals are selected based on a preprogrammed priority value, and an importance value is used to determine the impact of the goal's success or failure. The Tok agent uses a procedural reasoning system which has a library of plans. Every goal has one or more predefined plans that can be used to achieve it. These plans are sets of subgoals and actions, and have testable preconditions to check whether a plan could apply in the current world-state. All active plans and goals are stored in a structure called an

active plan tree (APT), which expands and contracts as goals or plans succeed or fail. Goals in the APT have success tests, expressions which are evaluated to test whether the goal has been reached. Plans have context conditions to test whether a plan can still succeed. After receiving sensory information, Hap modifies its active plan tree according to these tests. Information about the outcomes is sent to Em, which uses this to generate emotions based on the OCC model for emotions [OCC88].

3.2 FearNot!

FearNot! is an emergent drama system designed as an educational tool to teach children how to deal with bullying. In a typical FearNot! simulation, the user is shown a scene in which a character gets bullied, after which the user can give the victim advice on how to deal with the situation. The scene is then played again with the user's input. It is very important for the user to feel empathy with the characters, most notably the victim. Therefore the project focuses on creating believable characters. The architecture of a FearNot agent is shown in Figure 3.1.

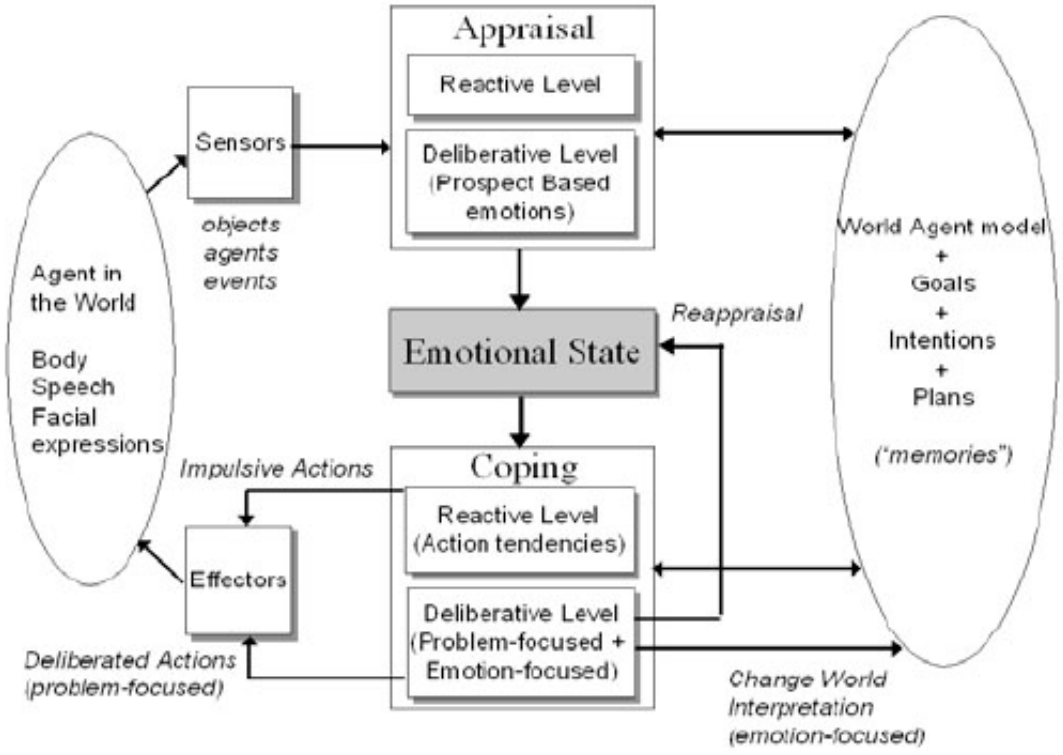

Figure 3.1: Architecture of a FearNot! character agent [ADP06]

In a separate project I have taken a more detailed look at FAtiMA, the agent architecture used by FearNot! [tB14]. The agents receive all perceptions, just like those in the VST. However the stories all take place in single locations such as a class room or a school yard. In this context the perception system is not really interesting as all participants are likely to get the same perceptions. How those perceptions are treated is more interesting. The agents use appraisal and coping mechanisms, based on the cognitive appraisal theory. Appraisal is the process that relates perceptions to the goals of the agent, generating emotions as a result. Coping is the process that deals with negative emotions, either by an internal adjustment or by taking external actions. These mechanisms are both divided into a reactive and a deliberative component. Like Oz, FearNot! uses an emotion system based on the OCC model for emotions [OCC88].

The reactive layer of appraisal uses a set of rules to generate emotions. Each rule has an event that triggers it and the OCC appraisal values that are affected by that event, e.g. praiseworthiness, desirability for others. The deliberative layer appraises events in relation to the character's goals. Emotion potentials of hope and fear are generated for every plan in memory, using the chance of success as well as the importance of success or failure. The resulting emotion is biased by the character's personality. The stronger emotion is used to guide planning.

The coping mechanism is similarly divided into reactive and deliberative components. The reactive layer uses a set of action rules. The agent's mental state is compared to that list and the rule triggered by the most intense emotion is chosen. If multiple matches come up the most specific one is chosen. The deliberative coping process first selects which of the agent's goals to attend to by picking the ones that generate the strongest emotions. Once a goal is chosen the most appropriate existing plan is selected or a new one is made, and brought into focus. This triggers the prospect based emotions: hope, fear and inter-goal fear. The last is fear that this plan threatens an interest goal, a state the character tries to maintain or avoid, e.g. a goal not to get hurt. An action is chosen and the cycle repeats.

3.3 Character-based storytelling

Another Interactive Storytelling system is presented in [CCM02]. The system makes stories in the sitcom genre, inspired by the television sitcom Friends. The stories it generates revolve around the main character Ross asking Rachel out on a date. The story is presented to the user through the Unreal 3d engine, which lets the user navigate the storyworld as a spectator. Additionally the user can interfere with the story by stealing objects from the world or by giving the characters advice through speech recognition. The system uses a character-based approach to the story, similar to the emergent approach used in the VST. The main plot results from the Hierarchical Task Network planner that defines how goals are decomposed into subgoals and possible plans to achieve them. At the bottom of these plan trees are terminal actions with preconditions similar to those in the VST. The variety of resulting stories is dependent on the interaction between the different characters' plans which differ due to random placement of characters at the start of the story. The preconditions for a terminal action may not hold, due to the activities of another character or user interference, causing action failure. In such a case the agent tries to repair the plan using action repair or situational reasoning. Action repair tries to restore the preconditions for the original action. For instance if Monica's presence in the room is preventing Ross from reading Rachel's diary he may wait for Monica to leave and continue his plan afterwards. Situated reasoning tries to respond appropriately to the current situation while trying to achieve a specific resulting state and avoiding undesirable results. In the above example, Ross could try to influence Monica to leave in order to continue his plan. This combines situated reasoning, dealing with the situation of Monica in the room, with action repair, restoring the precondition for the intended action. The paper doesn't specify whether an agent can find an object that has been moved or how it would do this. When the plan fails due to an object being stolen by the user, the agent will find the next applicable plan in its HTN and try that instead. Like the Oz system, this puts a lot of the creativity in the plan author's hands.

3.4 Game AI literature

Aside from dedicated storytelling systems, other fields of research were also examined. One closely related field is that of computer games. They also try to convey stories to the user in an interactive manner, using virtual characters in a virtual world. How are the non-player characters in computer games controlled? How do they perceive the world around them? Are they omniscient or do they only act on what they can see?

An interesting discussion is presented in [GvL02], concerning the use of virtual humans as participants (characters, IC) versus virtual humans as actors. The choice between participant and actor affects how agents perceive the world, how they communicate with each other and what

their goals are. Virtual participants should only perceive those parts of the world they can realistically perceive. Actors on the other hand should be omniscient. However, an actor should still be aware of the realistic bounds of his perception to remain believable. Virtual participants should only be allowed to communicate in a realistic manner, using a common language and following rules of proximity. Virtual actors don't need to adhere to such rules and can use additional means of communication hidden from the audience. Virtual participants only try to achieve the goals a normal human would pursue. Virtual actors should on the other hand appear to be trying to achieve those goals, while pursuing additional meta-goals that will increase the entertainment value for the user by trying to achieve a certain effect in the audience's mind. To be able to pursue such meta-goals, a virtual actor needs additional knowledge about how or when to elicit a certain response from the audience.

Another interesting article about the development of AI for opponents in computer games describes the concept of believable stupidity [Wes09]. In many games, a computer opponent has a vast advantage over a human player by the amount of calculations it can make. It could be evened out by reducing the amount of calculations the AI is allowed to make but this can result in the AI making incredibly stupid mistakes a human player would never make. A proposed alternative is to program the AI to set up situations that the player can exploit while still playing to the height of its ability in most cases. For instance an opponent in a chess game can be programmed to make a mistake that costs it a piece. After putting itself at a disadvantage, the AI continues to try and win the game. Lidén describes how opponents in a shooter game may be programmed to deliberately miss on the first shot, giving the player time to react as opposed to killing the player without warning. At the same time, the shots may be aimed at particular elements in the environment that will explode in a dramatic manner and provide an even more tense experience for the player[Lid04]. The important part is that the goal of the AI should not be to win but to provide an interesting game to the player.

3.5 Utility for the VST

3.5.1 Planning

The perception system used by Oz is quite versatile and enables some of the possibilities we would like to have in the VST. It allows for hiding information from agents as well as misinterpreting perceptions. Agents have an incomplete and possibly inaccurate view of the world, which makes it possible to deceive them. But how does the agent deal with the planning problem if he has an incomplete world perspective? This is one of the main problems the VST will encounter. An advantage Hap has over the planner used in the VST is that it doesn't have to make plans, only test which ones are applicable. This places a considerable responsibility in the hands of the author of the plan libraries. The agents are limited in their behaviour to what the author has given them. Also as the storyworlds get a bit larger and more complex, writing plan libraries will be a considerable amount of extra work. The plan library can be used to author very specific behaviour but prevents the agents from coming up with radically new behaviour. The same problem applies to the HTN system used by Cavazza's system. Emergent behaviour is one of the reasons a partial order planner was chosen for the VST [Kru07]. Scrapping the partial order planner and making a new one is not really an option for this project as it would be enough work to write a separate thesis.

The problem with the planner in the VST is that it tries to make a complete plan from start to finish, which will fail if crucial knowledge is missing. Due to the closed world assumption, any fact not explicitly known to be true is assumed to be false. If the location of an object is unknown, then it has no location. This makes the planner conclude that a plan to get that object is impossible. Hap has the ability to decompose plans into subgoals, filling in parts of the plan as needed. The agent can work on completing one subgoal while the information needed for a later subgoal is still missing. For instance, when Lyotard, the cat in the simulation, becomes hungry, he

has a number of strategies to find food. He can try the most likely place, his bowl, he can meow for attention or he can try a place where he remembers seeing food. He assumes that by executing one of these behaviours, he will find the location of the food, fulfilling the preconditions of the eat action. The partial order planner used by the VST does not make use of decompositions, only primitive actions.

3.5.2 Assumptions

What might be possible for the VST is to allow the planner to make assumptions about the location of an object similar to how Lyotard assumes he will find food in the most likely place. If a character has any belief about the location of an object, the planner would be able to make a plan that can be executed. It may turn out the food is not there, but if this is the case, new assumptions can be made. Unlike planning systems in other application domains, occasional plan failure is not a big problem in a storytelling context. Characters in stories regularly run into situations that don't turn out exactly as planned. How they deal with the problem is often an important part of the story. A key question will be how to make those assumptions believable, and additionally, how they could be used to steer the story in an interesting direction. One of the things the Oz project does is to use past knowledge in combination with current sensory information. The agents in the VST have an episodic memory which stores all past beliefs, perceptions, goals and actions. Up to now, not much was done with this since the current world knowledge was always correct and up to date. But when current knowledge no longer suffices, past beliefs can be used to provide useful information on where to look.

As mentioned, the perception system of FearNot is not very helpful for this project due to the single location settings it uses. Currently, VST characters don't have an emotional system. Once they do, perceptions will play a central role in it. Getting too many perceptions simultaneously will cause too many emotions which may lead to characters that appear inconsistent. As suggested in 2.5.3 an emotion system could also be used to influence perceptions.

Both FearNot and Character-based Storytelling allow for user interactions and suggestions. Having the user interact with the characters might be an option to assist with the planning problem. It brings to mind the concept of a Dutch childrens tv show "De film van ome Willem" in which a cast of actors plays out a story in front of a live studio audience. The characters will regularly turn towards the audience to ask them a question or to tell them not to betray a secret. In the interactive version of the VST a similar method could be used to have the user make suggestions to the characters instead of directly controlling the characters actions, to achieve a less direct form of control. For instance if O'Malley asks the user where Scurvy is, the user could then click on a location to make O'Malley look there. This can't be done in the current implementation however as the interactive interface was made after this project was already well underway.

3.5.3 Character versus actor

The characters in the VST are a mixture of actors and participants. They try to achieve their goals in-character like participants but also have out-of-character elements. They have the omniscient world knowledge that an actor should have, but currently no awareness of the realistic bounds of their perception. The actor and the character share the same knowledge base. Gordon and Van Lent [GvL02] state that "limiting a virtual actor to realistic knowledge and information effectively turns it into a virtual participant." For the VST actors one problem that would result from such limited knowledge is that they could no longer use OOC operators correctly. An actor could get stuck trying to frame something that already exists but the character hasn't seen yet. The world agent would reject the operator, but the actor would have no way of telling why it was rejected and try again. In order to use these operators, the actor will need full and accurate world knowledge. Instead of replacing this omniscient knowledge base with a realistic one, I propose

to split actor knowledge and character knowledge into separate knowledge bases. By doing this the actor's knowledge can be used to guide the character and perform OOC behaviour. At the same time the character's limited knowledge base can be used to improve believability. Instead of reducing the agent's knowledge, it will be increased. If this separation is made then the plot agent will have to keep sending perceptions to all the actors to keep their knowledge up to date. In that case it may be best to leave it up to the actor to decide what perceptions to pass on to the character, since the actor will know what the character should be able to see.

Like opponents in a computer game the goal of the actors in the VST should not be to reach their character's goal, but to create an interesting story. In the words of John Lennon: "Life is what happens to you while you're busy making other plans." This goes for characters in stories as well. Stories are not so much about characters reaching goals but about the conflicts they come across while trying to reach those goals. If the characters have only their goals to guide them, they will not seek out conflict because it is not asked of them and it would only impede on reaching their goals. Characters that have directly opposing goals may still get in each other's way. But a character will not take a wrong turn and walk into a trap. A storyworld author may provide all the elements required for a conflict, but if those elements don't come together in the story, there will be no conflict. Similar to the explosive elements in a shooter environment, a storyworld may have opportunities for events that, while interesting for the audience, are not in the interest of any of the characters' goals. If the actor knows about these elements, he can direct the character towards them, instead of directly towards the character's goal. At a moment when a character no longer knows where to go, the actor has a perfect opportunity to steer the character towards conflict without limiting the character's autonomy. Of course the story will still not be interesting if it is not believable so the actor will have to keep in mind what the character does know. However, by making the actor direct the character in a false direction, the character can be steered towards the conflict and be forced to deal with it. Previously, deceiving characters in such a manner would not be possible because they would know the truth and choose the easiest and shortest plan. By ensuring such direction only takes place when the character does not know exactly where to go, the chances of forcing the character into incoherent behaviour should be minimal.

It can be argued that guiding the characters towards conflict would reduce the emergent nature of the story by forcing it in a particular direction, but if this means that fewer uninteresting choices are made, that is not really a problem. Eventually the agent will discover the mistake and emergent behaviour will take over again to deal with the situation. If the actor has more options for conflicts to choose from, then it will add a different source for emergence, story goals instead of character goals. Additionally, an author would not put a lot of time writing possibilities for conflict in a storyworld to find out they don't take place because the opportunities never arise. In most stories a character has to make such a mistake to even create the basis for a story. Imagine a horror film in which the main character makes the wise and completely believable decision not to enter the creepy looking house. It would be a very short film. The audience has to suspend their disbelief about the main character's mistake to enter the building in order for an entertaining story to begin. Similarly, the audience should accept the fact that a character "accidentally" walks straight into a conflict if the conflict makes for an entertaining story. Whether the resulting story truly will be more interesting depends on many other factors, including the creativity of the author of the storyworld. But by allowing the characters to be directed towards the authored conflicts, an important step will be made.

3.6 Conclusion

None of the examined systems provide a solution for dealing with planning when characters don't have all the information they need, short of using a different planning system. Instead of making an entirely new planner, we will use an approach that fits in better with the existing planner, by making use of a new assumption operator. This operator will work similarly to

other existing operators making it easy to integrate with the system. It will allow a character to make assumptions about missing facts by making use of past knowledge which is stored in a character's episodic memory. Even though the information provided by the operator may be false, it will allow characters to plan with it and continue trying to achieve their goals instead of giving up.

To support both the actor and character roles that character agents in the VST fulfill, the knowledge base will be divided into in-character and out-of-character knowledge. The in-character part will be used to improve the believability of characters while the out-of-character part will be used to perform actor functionality such as improvisation and goal justification. Apart from that, the actor knowledge will also be used in assumption operators to guide characters towards interesting situations that they would have avoided if they had full knowledge of the world in-character. This way, the assumption operators will provide a new method of plot control that would not be possible with omniscient characters.

4 A new perception system

In Chapter 2 we have seen how perceptions originate and how they end up in the character's knowledge base. In Chapter 3 I argued that limiting this knowledge base to in-character knowledge only would break the actor's ability to use out-of-character operators. Instead I proposed to split the knowledge base into separate parts for the actor component and the character component. Plans the character makes can then be based on IC-knowledge, whereas the actor's OOC behaviour will be based on the full and accurate knowledge it requires. A number of steps has to be taken in order to achieve this. First of all the knowledge base has to be split into IC and OOC parts. This can be achieved by using named graphs to separate IC and OOC knowledge into different graphs. SWI-Prolog provides functionality for this but up to now it has not been fully used in the VST. Secondly, a set of visibility rules has to be written as discussed in Chapter 2. These rules then have to be applied to incoming perceptions to decide which ones to add to IC knowledge. Additionally, a way has to be added for a character to still get these perceptions at a later stage when they do become visible. Once the characters are keeping proper track of what they can see and know, they can be made to use only that knowledge for IC plans. This requires adapting the planner and underlying OWL reasoning to make use of the IC and OOC graphs. Finally, the characters' initial knowledge of the world will have to be set up. In this chapter I will describe how these steps have been implemented, resulting in a system where characters only act on what they have seen. Some initial deception is possible by giving the characters false beliefs at the start of the story. Another result is that characters get stuck when they don't know where their goal is, as was to be expected. How that problem is dealt with will be described in the next chapter.

4.1 The knowledge base

As mentioned before, named graphs can be used to separate the character's knowledge into different categories. In a way this was already being used in the fabula module. The main fabula graph stores all information about received and produced fabula elements such as type, what character they belong to, time and also the causalities between elements. Each element then points to a named graph that holds the actual contents of that element. However these graphs were not stored in the RDF database like the world state. Instead the (Subject, Predicate, Object, Graph) quads were asserted directly in prolog. The downside of this is that it bypasses the OWL reasoning layer. If we want to use OWL reasoning on the character's memories we will have to write the quads into the RDF database. This means that besides IC and OOC knowledge, the RDF database will also contain fabula quads. This brings us to the question of how to split up the knowledge base.

The most obvious separation is of course the one between IC knowledge and OOC knowledge. But where do they differ? Some things will remain the same no matter what happens in the story. Most notably, this is true for all the ontology information. This is all core knowledge about the concepts of the world, not its actual state. A character should be able to access this type of information either IC or OOC. The differences will be in the actual storyworld instance. So we only need to keep a separate copy of that in the IC and OOC graphs. A point could be made that some parts of the storyworld setting will also be static like for instance the location of a mountain. However making a separation between static and dynamic setting elements would

only complicate the authoring of the storyworld. Dynamic facts would all have to be stored in separate files from the static facts, so they can be loaded into separate graphs. It also means an author has to determine ahead of time what is allowed to change in the storyworld. In one story a ship could be defined as static as there are no actions implemented to sail the ship. A subsequent update could include new actions that allow moving or destroying the ship, which would mean the ship has to be moved from static to dynamic facts. It does not seem worth the trouble to make this distinction. This leaves us with 4 main graphs, namely core, IC, OOC and the previously mentioned fabula graph, which will still refer to subgraphs for the fabula elements.

Whenever a new agent is started up, the ontologies and setting files are loaded into the knowledge base. This is done at the RationalAgent superclass so it is the same for all agent types. At this point there is no need for IC knowledge yet as the agent may not even be a character agent, and even if it is, it has no role in the story yet. So at this point we load the ontologies into the core graph and the setting into the OOC graph. This means that the plot agent and the world agent also store their world in their OOC graph although they do not have an IC graph. How to set up the character's initial IC knowledge will be discussed further on. To load the files into the correct graphs, some work had to be done, which is described in Appendix A.1. The agents also need to write incoming knowledge into the correct graphs. Details about that process can be found in Appendix A.2. For the character agents to determine which facts should be added to IC knowledge, perception rules need to be added.

4.2 Visibility rules

In this project, the main goal was to limit perceptions to be able to hide information from characters. For this reason, the choice was made to focus first on visual perceptions. The other senses would mostly expand the amount of knowledge a character can receive which is of a lesser priority for this project. An approach was chosen that can be expanded upon in the future and can be made as complex as is needed for the storyworld. As storyworlds become larger and more complex, other senses or sources of information will get a more important role. Characters will need more ways to fill in the things they don't know. For now we ignore the fact that certain properties may not be visible properties. If more complexity is needed, properties could be assigned to a specific sense and use corresponding perception rules. Inference schemas could then be used to translate information from one sense into (possibly false) information for another sense. If a character could then plan for another character to have such a false inference, more complicated deception becomes a possibility. For instance, the wolf may plan that the little goat will mistake him for grandma goat and let him in the house, if he wears the proper disguise and softens his voice. Implementing such a scenario was out of reach for this project but in the future this may become a possibility.

An initial idea for visibility rules was described in Chapter 2. By making use of existing path relations between locations, we can determine adjacent locations of which the character might reasonably have a view. The advantage of using the path relations as a base for visibility is that if a path has a door, we can also check whether it is open or not. The rules work in three steps. The first step checks whether an object is at a location, either directly or indirectly, i.e being held by someone. The second step checks if an object is visible from a location making use of the first step to determine if an object is at a location. Objects at a location can be seen from that location. Objects at other locations can be seen if there is a direct path to that location, and if the path has a door, it is open. Objects in a container can be seen from the container's location if it is open.

Doors require a special case because they are not at a location. Doors are described as a part of the path relations between locations and are thus indirectly connected to a location. When a character walks from location A to location B, he is never really at the path. Instead, the character is removed from location A and added to location B when the walk action is completed. This means an additional rule is required to let the character see the state of a door. During one of the

first tests, this rule was not added yet and a character tried to open a door repeatedly. He did not see the result of his action, the door was opened on the first try but the character believed it was still closed. The final step checks if an object is visible for a character. This is true if the character is at a location, the object is visible from that location and the object is not hidden. Hidden is a new object property that was added to override the visibility rules.

These rules are quite basic and don't account for a lot of things yet but they can be easily expanded upon. For instance, currently there is no rule for windows as there are none in my storyworld. All that needs to be added is a rule that says location a is visible from location b if there is a window that connects them. Similarly, the rules assume all paths are short and straight so you can see down them. By expanding the ontology to make a distinction between short straight paths and long or winding paths, the rules can be changed to only allow viewing down straight paths. If the story allows characters to be blind or otherwise unable to see, these conditions can be added in the final step just like the hidden property.

If any concepts are added that say an object is at a location indirectly (not using the swc:at property) and it should be visible in that location an extra case has to be added to the rule ObjectAtLocation. The other visibility rules will then pick it up as normal. For instance, when Scurvy is caught by O'Malley, he is no longer at his current location but is instead being held by O'Malley who is still at that location. Since Scurvy is not a portable object the pick up action does not apply to him, so a new property is used to describe being caught. An additional rule is added that specifies that you are at a location if you are caught by someone and that person is at that location. This allows Scurvy to still see and be seen while caught.

This shows that visibility rules are an important part of the storyworld description. The responsibility of keeping them up to date with the storyworld is left to the author. On one hand this is a downside as it requires more work from the author. On the other hand it does allow for more flexibility. If an author wants to add magical crystals that allow a character to see distant worlds, they can. It would not be possible to simply capture such creative solutions in a general system that works for all storyworlds.

The rules can be used in schema preconditions. For example, the goal catchrat has preconditions that the agens is a cat, the patiens is a rat, and the patiens is visible for the agens. As a result, O'Malley does not take on this goal until he sees Scurvy. A downside of using this approach is that rules are marked as unplannable by the planner. During planning, the planner tries to unify the effects of available operators with the precondition it is trying to verify. Effects of an operator can only be facts, not rules. Rules can take very different shapes internally and there is no general way for the planner to find steps to solve them. If it encounters a rule in a precondition, it asks Prolog whether the rule has any solutions in the current world state. If there are none, the planner determines that the precondition is unplannable and will backtrack to try and find another plan.

As a result, the planner will not be able to find a plan to justify this goal. In other words, the planner will not make a plan that will result in O'Malley seeing Scurvy, so that he can adopt this goal. In this case that may be exactly as desired, delaying the goal until Scurvy triggers it. If however the precondition were used for an action to shoot someone, that would make it impossible for the planner to make a plan containing that action unless the target is currently visible. This is the problem with using rules in preconditions. On one hand they allow to make queries that are more than just a conjunction of true and false facts but on the other hand they make it impossible for the planner to satisfy them so they should be used with care. They can be safely used if the condition is not meant to be plannable. They can also be used to query things that the character has no influence on. For the shoot action, an alternative would be to use the visibleFrom rule, e.g. (Target visibleFrom LocationA). Prolog will provide any location for which this rule is true. Combine this with a fact (Shooter at LocationA) for which the planner can add the required plan steps. This way the action is plannable in more situations. It will still not work

if an action is required to make the visibleFrom rule true, such as opening a door.

4.3 First test of the visibility rules

As an initial test of the visibility rules, O'Malley was given a goal to catch Scurvy with a precondition that he can see Scurvy. Image 4.1 shows the layout of a pirate ship which was used as a basis for the storyworld. A few details were left out of the storyworld for simplicity's sake as well as to keep the planner from slowing down too much. The complexity of plans increases rapidly with the size of the storyworld and the resulting length of plans. At the rear of the ship is the poop deck (official naval terminology) with a ladder to the main deck. From there a door leads to the captain's cabin and a hatch leads to the gun deck below. The gun deck has passages to the crew's quarters and the galley on either side, as well as another hatch down to the hold. From the hold there's a door to the brig. Image 4.2 shows the basic map of the ship, the lines depict the paths between areas of the ship.

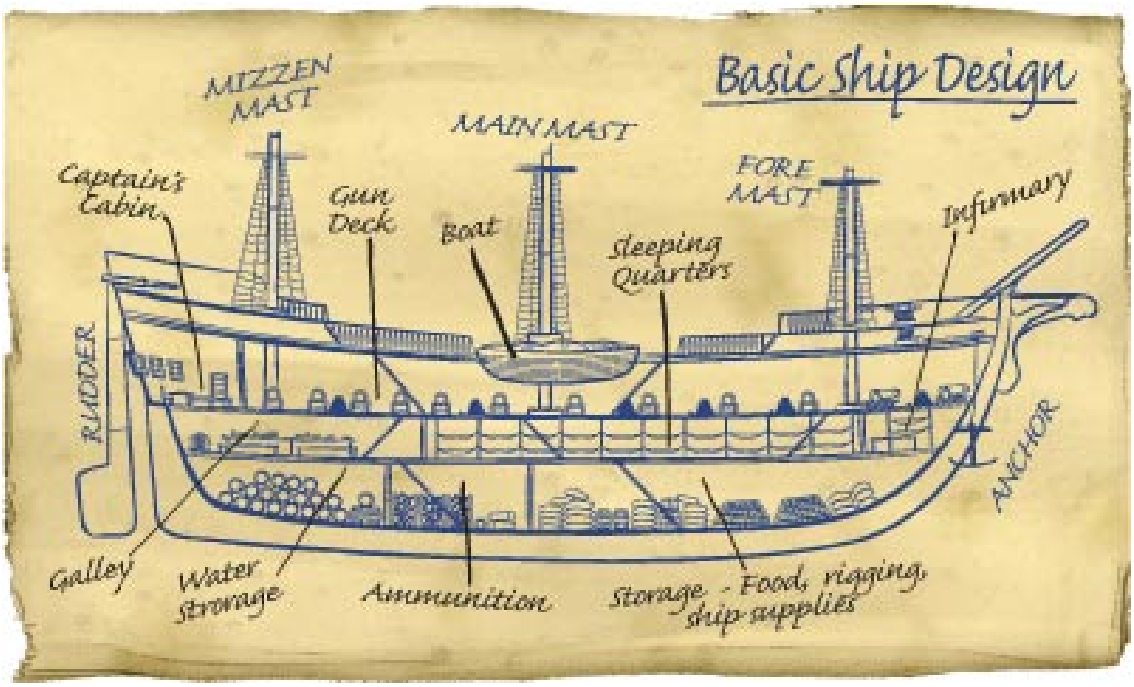

Figure 4.1: A basic pirate ship layout after which the ship in the story is modeled

At the start of the story, Scurvy is on the poop deck and O'Malley is in the hold, where the cheese is as well. The characters still act on their OOC knowledge because at this point, incoming character perceptions are not yet filtered by the visibility rules. Scurvy knows exactly where to go and plans to get the cheese from the hold. O'Malley in the meantime cannot justify the goal to catch Scurvy yet, as no plan can be made that fulfills the visibility precondition. As Scurvy gets to the gundeck, O'Malley can see Scurvy through the open hatch and takes up the goal to catch him. Scurvy has not been given a goal to evade O'Malley so he continues on to the hold and grabs the cheese. There is some odd behaviour at the doorway, because O'Malley has already decided to go to the gundeck to catch Scurvy. They pass each other without noticing and O'Malley revises his plan and goes back. This behaviour can be explained by the fact that the characters don't anticipate what the other characters are doing. They also do not cancel an action they have already started even when it is not needed for the plan anymore. The actions are also discreet, and don't describe a halfway-state. There is no point in the middle, where the characters meet in the doorway. These are interesting problems but fixing them is beyond the scope of my project. In a subsequent run of the story, the hatch to the hold is closed. This time, O'Malley

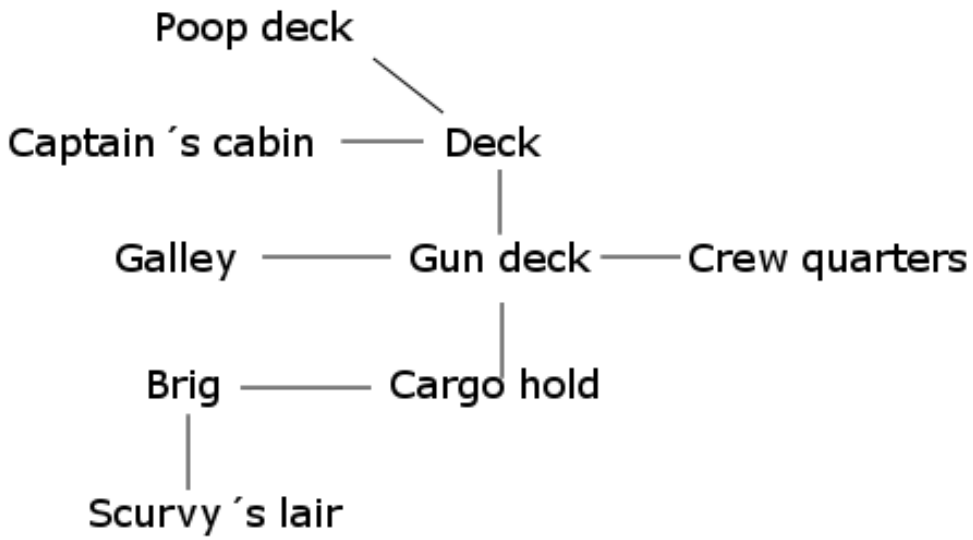

Figure 4.2: Schematic map of the ship in the storyworld

doesn't react until Scurvy opens the hatch. During these tests the visibility rules appear to work as intended. The next step is to apply them to incoming perceptions to determine which triples to add to the IC graph. This is done in the interpretationmanager as described in the next section.

4.4 An Improved InterpretationManager

Whenever the CharacterAgent is informed of a new perception, it is handed to the InterpretationManager to to be processed. In the old BasicInterpretationManager, all new perceptions were directly translated into beliefs. A new version of this module was created, the ImprovedInterpretationManager. It is here that incoming perceptions are turned into beliefs, so here is where the perception rules have to be applied.

The first thing a character should do with a new perception is handle it OOC so the actor's knowledge stays up to date. The contents of the perception are translated to a new beliefelement and asserted to the OOC graph. Once this is done, the perception is analysed. Perceptions can be one of two kinds. The first kind contains the actual results of an action or event, the triples that become true or false. The second kind contains the fabula description of the action or event. If the perception contains a triple, then the visibility of this triple depends on the visibility of its subject. An action is visible for a character if the character performing that action is visible. If either of these tests return true, then the perception should be processed IC. A fabula-element is created for the perception to be sent to the plot agent and the character's episodic memory. The same thing is done for the beliefs that follow from the perception, and a causal link is created between them. Finally the belief is asserted to the IC graph.

If the test returns that the object or character is not visible, the character should not get this belief IC. What should be done with perceptions that the character could not see? We could either discard them or store them for later. This becomes important when the character reaches a point where he can see the thing he missed before. Nothing prompts the character to check his surroundings and see if his beliefs are correct because they always were correct at all times. This is no longer the case, so the character needs a way to detect these changes as they become visible. If we keep track of things the character missed, it becomes much easier to re-test visibility later.

However if multiple actions in a row get missed, some of them may reverse triples added by a previous action. For instance if Scurvy walks from the poop deck to the deck and then to the gun deck, he is added to the deck at one point and removed later. If two such opposite effects are missed, nothing changed compared to what the character already knew. In that case they can both be discarded. The remaining triples are those things a character believes but are no longer true and those things that are true but the character has not seen yet. We store these in a list and make sure it has no opposite triples as described.

Whenever a new visible perception is received, these missed triples are re-tested for visibility. There are two cases that have to be tested for. The first case is triples whose subject has become visible due to the effects of the new perception. This includes true and false triples. For instance if O'Malley sees that Scurvy is at the deck, this means he will also conclude that Scurvy is no longer in the galley. The second case involves triples that have to be removed because they are no longer true. For instance, Scurvy believes the cheese is in the galley. He arrives there and the missed triples are re-tested. The cheese is not there so the visibility check returns false and the missed triple is skipped. Scurvy will keep on believing the cheese is there and try to pick it up. A character should discover such a false belief when he can see it is no longer true. To detect this, visibility is tested in both the IC and OOC graphs. If the character believes the object is visible but the OOC graph states that it is not, the OOC graph is right and the belief should be dropped. If a character believes an object is not hidden, but it has since become hidden, the same test is applied. Otherwise the character would believe the object is still visible and just ignore the fact that it is hidden. If any of these tests return true, a new belief is created for the triple as well as a causal link from the perception that caused it. Finally, these beliefs are passed to the reinterpret step, which is meant to derive new beliefs through a set of rules but does nothing yet as no rules have been implemented.

In order to monitor whether all the perceptions were being added properly and at the right time I implemented some debugging functions. These are described in Appendix B.

4.5 The planner and OWL reasoning module

With the perception rules implemented and applied to incoming perceptions, characters are properly keeping track of what they can see. However, they are still acting on all their knowledge including IC and OOC knowledge, but also episodic memory. The next step is to update the planner and underlying OWL module so that they will only query the right graphs.

The planner already keeps track of whether a plan is in character or out of character through the GoalInCharacter argument. If we pass this on to the lower layers of the planner we can use it to select whether to validate conditions in the IC or OOC graph. The question is where in the plan process the graph argument needs to be used. When fulfilling open preconditions, the planner chooses either a previous step in the plan, a new operator or the start state. If a previous step in the plan was chosen we can assume this step was added correctly and its preconditions are met using the correct graphs. If a new action is chosen, this will add preconditions that will also have to be achieved using the same methods. If the start state is chosen, then the condition has to be true (or false if it's a negative condition) in the current world state. In this case the conditions have to be validated in the correct graph depending on what type of goal the plan is for. If this is done, the rest of the plan will use the correct graph by recursion. The chooseStart rule of the planner has been altered to include the GoalInCharacter argument, and based on this, it chooses whether a condition should be validated IC or OOC.

The planner passes the condition along with the graph argument to schema management to be validated. There the condition is split into its subconditions, and each of those has to be queried in the correct graph. In some cases this is not the graph that was passed on by the planner. Subconditions can be facts, fabula or rules, and each of those requires different treatment of the graph argument. The choice was made to allow fact conditions that specify in what graph

they should be checked, regardless of whether the plan is IC or OOC. That way, an OOC plan can query whether the character knows something IC. The other way around, an IC plan can check whether something exists, regardless of whether the character knows about it. If the graph argument is not specified in the schema condition, then the graph passed on by the planner is used. For comparison, if a schema states:

```
condition(true(Fact(scs:Scurvy, swc:at, scs:oDeck_1)))
```

The condition will be validated in either the IC or OOC graph depending on the goal. If instead the condition states:

```
condition(true(Fact(scs:Scurvy, swc:at, scs:oDeck_1, ooc )))
```

This condition will always be validated in the OOC graph. In other words this forces the planner to check whether Scurvy actually is at the deck, regardless of what the planning character believes IC, even if the plan was IC. Using this approach, we could change the preconditions of O'Malley's goal to catch Scurvy to require that O'Malley knows Scurvy is at a certain location IC. In the test, the precondition for O'Malley's goal was that Scurvy would be visible. This makes a difference in when O'Malley will adopt the goal. Using IC beliefs instead of visibility rules will make O'Malley start to hunt Scurvy right away if he knows where Scurvy is at the start of the story. If he does not know, the knowledge will be added as soon as he sees Scurvy, at which point he will adopt the goal. How to set up these different starting scenarios is explained in the next section. During testing this condition turned out to be impractical because of the way the goal manager adopts new goals. It makes a new goal whenever Scurvy is at a different location because according to its comparison, it is a different goal that has not been adopted yet. The basic idea of the actor querying character knowledge and vice versa does work however and should be useful in other situations. A downside of using the graph argument in a precondition is that this makes the condition unplannable, just like using rules. This is because the planner matches conditions with effects and effects don't specify the graph argument as this is filled in at the time a perception is received by the agent. This makes such a precondition only useful for querying the state of the story at the time of planning and not for conditions that may become true by some kind of planning. This should be kept in mind when using such conditions in preconditions of actions. When querying whether a path exists as a precondition for a walk action, this would be no problem because regardless of the plan, the path will exist. When querying whether a door is open however, this would prevent finding a plan that makes the door open.

Unlike facts, rules are not validated directly in the rdf database. Some rules indirectly query facts, and in the new system it will be important to specify in which graph to look. However if we wanted to pass the graph argument from the planner to a rule automatically, like we do with facts, all existing rules would have to be rewritten to four argument versions, including the OWL rules. Not all rules need a graph argument, a lot of the OWL rules are even meant to look at multiple graphs. For example, if we want to check whether cheese is a portable object, the OWL reasoner checks what type cheese is and then whether that type is a subtype of portable object. The type of cheese is located in the IC and OOC graphs but the ontology information is contained in the core graph. This was solved by allowing a rule to have either three or four arguments, but the fourth argument is not automatically copied from the planner. If a rule has four arguments, like the new visibility rules, then the fourth argument has to be included in the schema condition. If the rule has to query character or actor knowledge the fourth argument can be used as a graph

argument. This assumes that if a rule in a schema condition needs the graph argument, it is not dependent on whether the plan it is used in is IC or OOC. If the graph argument is not needed a rule could also use the fourth argument for different purposes. For instance a rule to determine if there is a path from A to B could use the fourth argument to specify a maximum length. This might be useful for implementing audibility rules at some point.

If a condition checks fabula elements, the IC or OOC argument given by the planner is also disregarded because fabula is stored in separate fabula graphs. For the character agents, these are all episodic memory graphs, which are IC. In the current implementation, the actor does not store fabula of its own yet, although this may be useful in the future to allow the actor to store and query information about the internal workings of other characters.

After determining which graph to use, the query is passed on to the OWL reasoning module. This makes use of the relations between properties defined in the ontologies to try and unify the query with a triple in the RDF knowledge base. To achieve this it makes use of the rdf_has rule which will also find a triple if it has a predicate that is a subproperty of the queried predicate. An equivalent function with a graph argument does not exist, and would not work because these facts could be spread across multiple graphs. A version of rdf_has with four arguments does exist however. This previously unused fourth argument is used to return the triple's property, which as explained, could be a subproperty of the queried property. For example, cheese has type food, which is a subtype of portable object. So if we query whether cheese is a portable object, it will return true and the fourth argument will return food. By combining this with the subject and object from the query, we can reconstruct the triple that rdf_has found. But there is no way to tell in what graph it was found. So we need to chech whether the triple (cheese type food) exists in the correct graph.

4.6 Initial character knowledge

At the time a character agent is started, it has no role or identity yet. Only when the plot agent casts a new character does it get a role. At that time, the character should also get its initial knowledge of the world. If not, it will have an empty IC graph and be unable to plan. How should this knowledge be set up? The easiest approach would be to simply copy the OOC graph and start with a full knowledge of the world which will only start to differ from the truth as other characters act on it outside the character's view. A second approach would be to start with nothing except what is visible for the character from the starting location. However, this means the character knows nothing else of the world or its layout, making planning very difficult. The final option is to give each character a separate copy of the setting and allow it to differ from the truth. This option was chosen because it allows for the most flexibility. By changing a few things in the characters' initial knowledge, their behaviour can be manipulated without having to change too much to the actual setting.

To accomplish this, each character is given a personal subfolder in the setting folder. When the character gets a role, the prolog file from the corresponding subfolder is loaded, which contains the load instructions for the personal setting files. In these personal files, certain lines can be commented out or altered to hide facts from a character or give him false beliefs. This way we can make sure O'Malley does not know where Scurvy is at the start of the story and as a result, will not take up the goal to catch Scurvy until he first becomes visible. At the same time, it allows us to give Scurvy the false belief that the cheese is in the galley. After loading in these files, the interpretation module will need to be informed of the triples this character "missed", so the appropriate perceptions can be given when the character can see them. To accomplish this the OOC and IC graphs are compared to find a list of triples to add and remove and the result is sent to the interpretation manager to add to its list of missed perceptions.

4.7 Second test

With all the previously mentioned pieces in place, a few new tests were conducted. For the first run, Scurvy was made to believe the cheese was at the galley by altering this fact in his version of the setting. Scurvy proceeded to the gun deck, noticed the piece of cheese in the hold and grabbed it. O'Malley behaved just the same as in the previous test and chased Scurvy from the moment he became visible, catching him eventually in the hold.

Figure 4.3 shows a graphical representation of Scurvy's episodic memory after the story. The small squares marked P and IE are perceptions and beliefs respectively. The GetFood goal motivates all the actions. Scurvy starts by moving from the poop deck to the deck and opening the hatch down to the gun deck. The perception just after WalkFromToDoor (marked in yellow) is the perception just after Scurvy arrives at the gun deck. The internal element above that (also marked in yellow) is the belief that the cheese is at the hold, which enables the action to pick it up later on in the story. The unconnected actions in the lower right are O'Malley's actions that Scurvy has seen. They are unconnected because Scurvy does not know O'Malley's internal workings, the goals that motivate them or any other form of causality. There is also a failed HideWithFood goal that Scurvy couldn't complete.

For another test the starting setup was changed to make Scurvy believe the cheese was in the captain's cabin. Once again Scurvy proceeded to where he thought the cheese was. After opening the door to the captain's cabin, he noticed the cheese was not there and the story ended. As expected, Scurvy had no other idea where to look for the cheese and failed to make a plan to get it. He gave up and had no other goals to pursue, so he got caught by O'Malley.

4.8 Conclusion

This chapter describes the process of splitting the agents' knowledge into IC, OOC, core and fabula graphs as well as the implementation of visibility rules and applying them to incoming perceptions. It also shows how the planner is made to use the correct set of knowledge when making plans. The tests show that the characters now only use the IC knowledge to make plans. The final test shows that Scurvy can get into a situation where he no longer know where to find the cheese. The planner has no method to resolve this problem and the story ends in a rather disappointing way. Additionally the end is not very believable. Scurvy shouldn't just give up looking for the cheese just because it was not where it was believed to be. How can Scurvy be made to look for the cheese in a believable way, without outright telling him where it is? This problem will be explored in the next chapter.

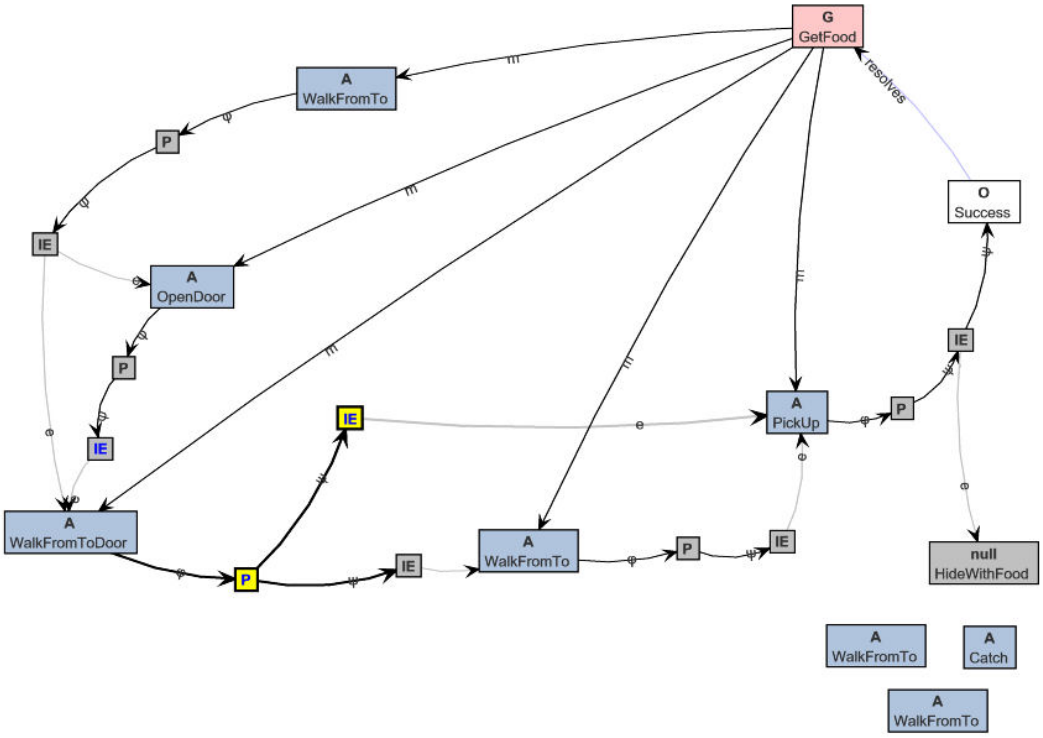

Figure 4.3: Scurvy's episodic memory after the story

5 Making assumptions

In the previous chapter the new perception system was introduced. The test runs showed that the story will still play out as normal if a character happens to come accross the object he needs while he still has a working plan. If however the character no longer knows where an object is the planner fails to make a plan. The character will abandon the goal and if there are no other possible goals, stop doing anything. That is not going to improve the resulting stories. Some way will have to be found to fill in the missing information.

There are many sources that could be thought of for giving the character more knowledge. We could expand the character's senses, adding sound or smell to provide additional ways for a character to receive a perception. But that would still not solve the problem. The character would either know exactly where to go, or none of the perception rules apply and the character stops moving. Social interactions could be used to share information between characters. But that would require a whole different layer of reasoning. What can be safely told to whom, when should a character lie, what are the social relations, enough questions to write a separate thesis on. What we would like to achieve is for the character to search for the object, and to illustrate the fact that the character doesn't know exactly where everything is it should be possible for a character to go the wrong way. This should be achieved in a way that works without making drastic changes to the surrounding systems. In chapter three the idea was raised that the planner could make a plan if it were allowed to make assumptions. If those could be captured in a schema just like other operators then it would be possible for the planner to use them without much modification. In this chapter that idea is explored further.

5.1 The problem

When trying to make a plan to pick up the cheese, one of the preconditions for picking it up is that Scurvy and the cheese are at the same location. Actually this is split into two parts, as follows:

```
condition(true, [
        fact(Patiens,     swc:at, Target),
        fact(Agens, swc:at, Target)
        ])
```

In this example, the patiens would be the piece of cheese and the agens Scurvy. Before the planner can make any plan to send Scurvy to the target location it will try to find a fact that fits this description, and if it can't find that, an operator that puts the cheese at the target location, otherwise there will be no location to send Scurvy to. But the only types of operators that will do that would be actions involving the cheese. And all of them require that Scurvy already has the cheese or at least knows where it is. What is needed is an operator that provides a location for the cheese, where the character had none before.

The effect of the assumption operator should be a fact that gives a location to the cheese in char-

acter. To the planner it doesn't matter whether the cheese actually is at that location, it can just be a guess. It will assume that the operator does what it says it does and create the rest of the plan to move to that location and pick up the cheese. During execution of the plan it will become clear whether the assumption was correct. If it was not, the character should get a perception that makes this clear. The character will re-evaluate his plan and conclude it is no longer applicable. A new plan will have to be made, possibly using another assumption.

Writing a separate assumption for every object a character can lose track of would soon become unmanagable for larger storyworlds, so a more general operator is needed. A schema for a general "AssumeObjectAtLocation" assumption would have three arguments: agens, patiens and location. The agens is the character who makes the assumption. The patiens is the object the assumption is about, the location is where the object is assumed to be. All of these will be subject to certain preconditions in order to keep the assumptions believable.

5.2 Preconditions

If we allowed the planner to make assumptions without proper preconditions, it would simply assume any object is at the character's location because this would provide the shortest plan. What's more, if the planner is allowed to make assumptions about an object at any time, it will assume the object is at a more convenient location even if the actual location is known but would result in a longer plan. A couple of rules are needed to make assumptions work in a believable manner. The first rule will put some general preconditions on the Location argument and the Object. If a character knows the location of an object, it should not be allowed to make any assumptions that contradict this knowledge. By making use of the ObjectAtLocation rule from the visibility rules, we can determine if the character has IC knowledge about the location of the object. If so, then that is the only location allowed for the assumption. Such an assumption will be left out of the plan because it adds nothing and the plan would be shorter without the assumption. If no IC location is found, then any location will be considered, as long as it is defined as a location or a subclass of location.

Another precondition is that the object does exist, in other words, that the object is at some location according to the OOC graph. This assures that a character no longer keeps searching for an object if it no longer exists. It is a trade-off where we have to give the character some information it couldn't know because otherwise there is no way for the character to find out. For example, if Scurvy was looking for the cheese but O'Malley ate it for some reason, there would be no way for Scurvy to discover the cheese no longer existed. Even if he got to the location where O'Malley ate it, the conclusion would be the cheese is not there, but it might be somewhere else. Scurvy would keep searching when in fact, his goal is no longer achievable. Ideally, there should be a method for the character to believably determine that an object no longer exists but that will have to be explored in future work.

The character's current location is excluded as a possibility. If the object was at the character's current location, he would have seen it, even if it was in an open container. This assumption does not cover closed containers or objects that have the hidden property. Those would require separate assumptions to give the character the idea to open the container or use a search action to reveal the hidden object. In the example stories those situations don't exists, so assumptions were not made for them. But even given these preconditions, the character will end up walking back and forth between two locations unless he runs into the object he is looking for by chance. That's still not a very believable behaviour. We need something more to make sure the character searches in a more believable manner.

In chapter three the idea of using past knowledge to guide the character was mentioned. The Oz system integrates current sensory information with past knowledge in its integrated sense model and uses the combined knowledge to select possible plans. The characters in the VST do a similar thing by storing beliefs and perceptions in episodic memory. If we make use of these memories,

we can find out where a character has been. Since these memories are already recorded with a time, we could even find out how long ago a character was there. To do so we would have to make characters explicitly store the current time each turn to ensure this information is up to date. Using this information, we can exclude a location as a believable search location if the character has been there less than a certain number of turns ago. How large a number of turns will be dependent on the size of the storyworld and the speed with which it changes. To achieve this, some new functions were required. First of all a new query function was added that takes a subject, predicate and object and finds the latest beliefgraph that contains this triple. Some additional utility functions were added for finding the time associated with a fabula-element and converting this to a value prolog can use in calculations. Using these tools the second rule was set up which tests whether a location is believable for a character to search. If a character has never been to the location, it passes the test. If the character has been there, then the time for the last belief that the character was there has to be more than 10 turns ago, otherwise the location is excluded.

Rule 1
Location1 is a proper search location for Object1
***if** the character doesn't know IC that Object1 is somewhere else*
***and** Location1 is a type of location*
***and** Object1 is somewhere in the storyworld*

Rule 2
Location1 is a believable search location for Object1
***if** character has no memory of being at Location1*
***or** The latest memory of the character being at Location1 is at least 10 turns ago*

Using the new rules, a test implementation of the "AssumeObjectAtLocation" operator was made. Each character was made to update the current time in their knowledge base at the start of each round. A temporary alteration to the planner allowed it to see assumptions as starting steps in a plan. This enabled quick testing of the assumption schemas without having to further process the assumption. The downside is that this way the character doesn't make any record of the assumptions as no fabula is generated yet. This will be necessary to communicate the assumption to the audience. In the final implementation this is achieved by implementing the assumption operators as internal element operators, as described in the next section.

During the test, Scurvy was once again set on the wrong path to the captain's cabin. Once again he opened the door to the captain's cabin and saw no cheese. However, this time he was allowed to make assumptions. His first assumption was that the cheese was in the captain's cabin, since he hadn't actually been there yet, so he proceeded to look there after all. Finding nothing, he made another assumption and went to look at the gun deck. As he arrived there he could see the cheese in the hold and the story ended as before, with Scurvy grabbing the cheese and O'Malley catching Scurvy. In another run, Scurvy was given no knowledge of where the cheese was at all. He proceeded to methodically inspect unvisited areas until he would reach the gun deck from where he could see the cheese, after which the story would end as before. During this test, there were no containers and the cheese could not be hidden. Given this simple environment, the assumption was used as expected and made Scurvy search for the cheese in a reasonably believable manner.

5.3 Internal element operators

In order to get the results of assumption operators to be logged in fabula, they will have to be implemented as proper operators. Instead of faking the result and allowing the character to act on it, the result will have to be asserted to the knowledge base and logged in fabula. Assumptions and guesses fall under the category of internal element operators. Instead of making a change in the actual world, they change the internal world representation of the character that makes them

only. The planner already includes internal element operators as plan steps. By implementing assumptions as internal element operators, they can be included in plans as normal operators. When the preconditions are met, the planner will return them for the character process to execute. With the new distinction between IC and OOC knowledge, some changes in the execution of internal element operators are also required. Since internal element operators don't change anything in the world, their effects should be IC only. The plot and world agents should not make any world changes. Fabula elements should be sent to the plot agent for the assumption as well as any beliefs that follow from it, so they can be translated into the resulting story. The choice was made to turn assumptions directly into beliefs.

Whenever an assumption is made, there is a chance that it was a false one. If this has happened, then the character must have a way to discover the mistake. A perception will have to be triggered as soon as the character is within visual range of the subject. Otherwise the character will end up repeatedly trying to act on the false knowledge. This is achieved by informing the interpretation module of any false assumptions so it can add them to its list of missed triples. This will cause the appropriate perception to trigger as normal.

With these changes, the character agent properly executes assumptions and adds fabula for them. It also retracts them if they prove to be false. However, assumptions are mostly filtered out of the fabula displayed on the agent's fabula panel. This is because uninteresting ends are filtered out. Since most of the assumptions end up being false, they do not directly connect to the pick up action. Eventually a perception triggers the belief that forms the basis for the pick up action. The walk actions have no connection of their own to the assumption or belief that the cheese is at the target location. There is no subgoal of finding the cheese which motivates these actions, just the goal to have the cheese. Internal elements are currently filtered out of the fabula if they have no outgoing edges that connect to the main storyline. Because of this it is sometimes difficult to follow a character's actions from just the filtered episodic memory display. A planner with decompositions would be better at describing the events making it easier for the narrator to translate them. Part of the goal to have the cheese would be to find out where it is. This subgoal would be decomposerd into various walk actions eventually resulting in a perception of the cheese's location. The narrator could use the higher level subgoal in its translation. Instead of a formulation like: Scurvy walked here and then he walked there, it would be: Scurvy searched for the cheese and finally found it in the cargo hold. For the planner itself this would also be more efficient because it would not have to plan the followup actions until the cheese was found. The assumptions are still recorded in fabula however and will show up if the knowledge base is saved to a textfile.

5.4 Conclusion

In this chapter, a general implementation of the assumption operator was discussed. This new operator type can be used by a character to make assumptions about the location of an object. This in turn allows the planner to make a plan using this information. If the assumption was correct, the character will find the object and can continue his plan. If not, a new assumption can be made, until the object is found. By using assumptions, Scurvy is able to find the cheese without knowing exactly where to look. During his search, he might encounter other things he was also unaware of. He may run into a conflict he was not expecting. But then again, he may also get lucky and guess right on the first try, avoiding any trouble. Whether or not the story is interesting is left to chance. Putting the characters and objects in the right locations may ensure that a conflict will start. But in a longer story, controlling the location of objects like that may not be possible.

In Chapter 3 the idea was mentioned to use actor knowledge to guide the character in interesting directions. Whenever a character wants to make an assumption, there is obviously something the character does not know. This is an ideal moment to steer the character in a direction he would

not willingly choose if he knew what was waiting. By making use of the character's ignorance, we can submit him to an unexpected form of deception. The character more or less deceives himself with help of some actor knowledge. The underlying idea is that an author usually does not send a character to a location without a reason. The character may find what he is after, or he may find something else, positive or negative. Ideally we would want to give the actors some sort of meta goals to help make the story more interesting. However that is outside the scope of this project. For now, a simpler, more direct approach will be used to give an impression of the possibilities. By making separate assumption schemas for each phase of the story, we can add additional preconditions to either introduce conflict or resolve it. In the next chapter we will apply this to Scurvy's domain to try and spice up the story a bit.

6 Authoring a story

In the previous chapters the perception system was tested using a simple incomplete story. Both characters only had a single goal and the story would always end with O'Malley capturing Scurvy. Even if Scurvy managed to get the cheese, he would no longer have any goals to pursue and O'Malley would capture him. In this chapter we will evaluate the system by fleshing out the story a little more. This also gives us an opportunity to take a closer look at the authoring process. First we will describe the story we would like to generate. After that we will discuss the new elements that we will have to add to the domain. After that we will describe some of these elements in more detail. Finally we will take a look at the story that is generated.

6.1 Expanding the story

In [Tom09] a method was devised to make a story follow Freytag's pyramid by adding conflicts leading up to the climax of the story and using resolution goals to lead the story towards a conclusion. A story consists of an exposition phase, a rising action phase which ends with a climax, a falling action phase in which the remaining conflicts are resolved and finally a dénouement. Some inference operators were added to help the plot agent to keep track of the story phase, so that it can suggest the right goals to the characters. Every plot thread has initial goals for the characters as well as resolution goals that will be assigned to the characters once they have finished their initial goals. We will make use of these techniques to improve Scurvy's behaviour after he has found the cheese.

In the new story, Scurvy will still want to obtain the cheese. We will not tell him where the cheese is, so he will have to search for it. To make sure the story is somewhat exciting, we would like to make sure that he runs into O'Malley which will trigger O'Malley's goal to catch Scurvy. At the same time, Scurvy will have a goal to escape O'Malley and avoid being captured. At some point, Scurvy will have to deal with O'Malley so that he can get the cheese without being captured. We add some methods for him to keep O'Malley distracted. As explained in Chapter 1, the inspiration for the setting comes partly from Tom and Jerry cartoons and this can be seen in the methods Scurvy will use to distract O'Malley. Just as Tom is sometimes scolded by his owner for the mess that results from Jerry's antics, O'Malley might get in trouble for messy situations on board the ship so he will try to avoid this. Physical injuries resulting from comical actions are also quite common in these cartoons so we will also make use of those. Once Scurvy has the cheese, he should take it to a safe place where he can eat it in peace, to prevent the ending we saw in the previous chapter.

6.1.1 New goals

For this new story, O'Malley's behaviour will mostly be the same as before. He will get a goal to catch Scurvy as soon as he can see him. He is given one new goal which is to swab a deck if it is slippery. This has a high priority for him because he does not want the captain to get angry by slipping on the deck. The captain is not in the story, he is just the motivation for this goal.

Scurvy also gets some new goals that will help him avoid being captured. Previously, Scurvy would only have the goal to get the cheese and therefore he would just ignore O'Malley's pres-

ence. To prevent this, we add an `escape` goal, which will make Scurvy flee once he sees O'Malley. To ensure that it takes precedence over the find cheese goal, we set it to a slightly higher urgency.

Once the escape goal has been triggered, Scurvy will try to run from O'Malley, but since they both move at the same speed, Scurvy will not be able to complete this goal. Eventually, he will either get backed into a corner or he will find the location of the cheese. If either of these happen, Scurvy will need a way to keep O'Malley busy. Otherwise he will not have enough time to get away or pick up the cheese. So for this purpose we will add a `distract` goal. Some new actions are added to help Scurvy complete this goal.

One problem with distracting a character is that characters don't respond to what the other characters are doing. This means that Scurvy will not realize that O'Malley is not going to follow him for a while when any of the methods of distraction have been used on him. To work around this we add a `distracted` property to inform Scurvy that he can go back to his other goals. Of course we would eventually like the characters to take note of the effects of other characters' actions and incorporate them into their own plans, or if necessary, try to prevent them from completing. This will be left for future work, for now we will use the workaround.

Once Scurvy has obtained the cheese, we want to make sure he gets to a safe place. For this purpose we have added the `hidewithfood` goal. This is also the resolution goal, which makes the story progress to the falling action phase.

6.1.2 Additions to storyworld

In order to give Scurvy a safe place to hide once he has the food, we add Scurvy's lair to the ship. It is connected to the brig by a new path. This path is of a special type, a RatTunnel, which is too small for O'Malley to fit through. We also add a cannonball to the gun deck, which will be used to distract O'Malley. There is also a bucket which contains soapy water which is placed on the poop deck. The complete description of the storyworld can be found in Appendix C.

6.1.3 New actions

Some new actions are added to help Scurvy achieve the escape goal. First of all there is `DropCannonBall`. This makes Scurvy drop a cannonball on O'Malley's foot. This gives O'Malley a sore foot, preventing him from following. Second, there is an action to kick over a bucket of soapy water, which will make the deck slippery. This will force O'Malley to swab the deck. In both cases, the distract goal will be completed. For O'Malley there is a `FixFoot` action, which causes him to dance around on one foot, screaming in pain, after which he can move normally again. There is also the `SwabDeck` action which he can use to clean a slippery deck. Finally, for Scurvy there is the `CrawlFromTo` action, which will let him use the tunnel to get to his lair.

6.1.4 Guiding the story

In order to make the story progress the way we want it to, we disable the original assumption operator. We add two new assumption operators in its place. The first makes Scurvy assume the cheese is at the location where O'Malley is, by using the actor's knowledge. This can be used early in the story to ensure that Scurvy will run into O'Malley and begin the conflict. The second assumption lets Scurvy assume the cheese is at the location where it actually is. This can be used at any point after O'Malley has been distracted at least once and thus the conflict has already taken place. So from then on the story will start to work towards a resolution. Initially we wanted to use the story phase to help decide which assumption to choose but this wouldn't work because of the way the storyphases are implemented. The falling action phase can only

begin after the hero has completed his initial goal, which for Scurvy is to obtain the cheese. Thus he wouldn't be able to use the assumption that leads him to the correct location until after he had found the cheese, when it is no longer needed.

6.2 Authoring

In this section we take a closer look at some of the goals and actions. How do they work together, what are the preconditions and success conditions for the goals or the effects for actions?

6.2.1 Goals

The goal to find food (shown in Table 6.1) has simple preconditions. The agens, the actor who takes this goal, has to be of type rat. The patiens has to be some type of food. The agens must not yet have the patiens. For this goal to succeed, the agens has to have the patiens. We keep it simple so it will only work for this story. It could be argued that a rat is also a type of food so that O'Malley's goal to catch Scurvy could be replaced with a `getFood` goal. But in that case, the fact that a rat is food would be conditional on whether you are a cat. This would only complicate matters. Furthermore, we want O'Malley to only start his `catchRat` goal when he sees Scurvy, whereas Scurvy has to start working towards his goal immediately in order to get the story started.

Table 6.1: The GetFood goal

GetFood	
Preconditions	
true	rule(Agens, owlr:typeOrSubType, scs:'Rat') rule(Patiens, owlr:typeOrSubType, scs:'Food')
false	fact(Agens, swc:has, Patiens)
Success conditions	
true	fact(Agens, swc:has, Patiens)

The escape goal (shown in Table 6.2) is a little more complicated. To escape O'Malley, Scurvy has to make sure he is not seen by O'Malley. Ideally, we would want to use a negation of the rule Scurvy `visibleFor` O'Malley. However as mentioned in Section 4.2 the planner has no method of planning actions to achieve this. When it encounters a rule it can only test whether it is already true and if not, the condition fails and the planner will deduce that it cannot make a plan to achieve the goal. Like the example with the shoot action, we can use the `visibleFrom` rule instead. Since a location being visible from another does not change except by closing a door, we can use that rule without running into an unplannable situation. The escape goal succeeds if Scurvy is at a location that is not `visibleFrom` O'Malley's location. Prolog will return a list of such locations and the planner can include actions to go to such a location.

The first two preconditions are that the agens is a rat and the patiens is a cat. The patiens has to be visible for the agens. Finally, the agens also has to be reachable for the patiens. This prevents Scurvy from selecting the goal again once he is safe in his lair. We also add a precondition that prevents Scurvy from taking this goal when O'Malley is distracted. At the same time we add a failure condition that makes Scurvy drop the goal when O'Malley is distracted, otherwise he would not go back to the `GetFood` goal which has a lower priority. Using a failure condition for this is not ideal but there is currently no other method of making a character drop a goal or lowering its priority when certain conditions have been met. There is also no method of using a disjunction for success conditions so that the escape goal can be completed by either escaping or distracting, except by using a rule, but that would lead to an unplannable goal again. The success conditions are split into three parts. The first will select any location. The second part ensures

that the location is not visible for O'Malley. The final subcondition is the plannable part which makes the planner devise a plan to get Scurvy to the selected location.

Table 6.2: The Escape goal

Escape	
Preconditions	
true	rule(Agens, owlr:typeOrSubType, scs:'Rat') rule(Patiens, owlr:typeOrSubType, scs:'Cat') rule(Patiens, swcr:visibleFor, Agens, ooc) rule(Agens, swcr:reachableFor, Patiens, ooc)
false	fact(Patiens, swc:hasAttribute, scs:distracted)
Success conditions	
true	rule(Target, owlr:typeOrSubType, swc:'Location')
false	rule(Target, swcr:visibleFor, Patiens, ooc)
true	fact(Agens, swc:at, Target)
Failure conditions	
true	fact(Patiens, swc:hasAttribute, scs:distracted)

The distract goal (shown in Table 6.3) has two possible situations in which it can be selected. The first is when Scurvy is cornered. The second is when Scurvy has discovered the cheese but is being chased by O'Malley. Instead of making two goal schemas, we make use of goal selection rules. These are schemas which list only the preconditions for when a goal can be selected. When selecting a goal, the character agent's goal selection strategy looks at the goal selection rules and picks those rules for which the preconditions hold. If no goal selection rules are specified in a domain, a simpler strategy is used which looks at the preconditions of a goal schema itself.

The first goal selection rule (shown in Table 6.4) has two preconditions to ensure the agens and patiens are a rat and a cat respectively. The third precondition specifies that Scurvy has to be cornered by O'Malley. This rule is true if Scurvy is at location A, O'Malley is at location B, B is directly adjacent to A and there is no location C such that C is not B and C is adjacent to A. In other words, if Scurvy is in a location and the only way out is past O'Malley. The final two preconditions are that Scurvy is reachable for O'Malley and O'Malley is not distracted. For the second goal selection rule (shown in Table 6.5), we replace the cornered condition by two lines that specify there is an object of type food and this object is visible for Scurvy. We also add a condition that Scurvy does not have the food yet. Both of these rules result in the same goal schema being adopted.

As with the escape goal, we add a failure condition to the distract goal that will make Scurvy drop the goal as soon as he has got the food and escaped to his lair. Otherwise he would get out and try to distract O'Malley after he had completed the goal to get to safety, which does not make sense.

The `HideWithFood` goal (shown in Table 6.6)is much simpler than the previous two. It requires that there is a rat, a piece of food and a rathole. The food must be in possession of the rat and the rat must not be at the rathole yet. The goal succeeds when the rat has reached the rathole and still has the food.

O'Malley's two goals are simple in comparison to Scurvy's. For the `CatchRat` goal(shown in Table 6.7), if there is a rat and a cat, the rat is visible for the cat and the rat has not been caught yet, the cat should adopt the goal to catch the rat. For the `CleanDeck` goal(shown in Table 6.8), if there is a deck and it is slippery, adopt the goal to make it not slippery.

A more interesting part about the `CleanDeck` goal is how it makes O'Malley distracted. The

Table 6.3: The Distract goal

Distract	
Preconditions	
true	rule(Agens, owlr:typeOrSubType, scs:'Rat') rule(Patiens, owlr:typeOrSubType, scs:'Cat') rule(Patiens, swcr:visibleFor, Agens, ooc)
false	fact(Patiens, swc:hasAttribute, scs:distracted)
Success conditions	
true	fact(Patiens, swc:hasAttribute, scs:distracted)
Failure conditions	
true	rule(Target, owlr:typeOrSubType, scs:'Food') fact(Agens, swc:has, Target) rule(Loc, owlr:typeOrSubType, scs:'Rathole') fact(Agens, swc:at, Loc)

Table 6.4: First goal selection rule for Distract

DistractRule	
Preconditions	
true	rule(Agens, owlr:typeOrSubType, scs:'Rat') rule(Patiens, owlr:typeOrSubType, scs:'Cat') rule(Agens, scsr:corneredBy, Patiens) rule(Agens, swcr:reachableFor, Patiens, ooc)
false	fact(Patiens, swc:hasAttribute, scs:distracted)
Selected Goal (Goal)	
	goal_schema(Goal) schema_type(Goal, scs:'Distract') schema_agens(Goal, Agens) schema_patiens(Goal,Patiens)

`KickBucket` action, unlike `DropCannonball`, by itself does not make O'Malley distracted. Its effect is that the deck becomes slippery, but Scurvy still has to derive the fact that this will distract O'Malley. Otherwise, the planner has no reason to select this action. To reach this effect, we make use of reactive actions and expectation schemas.

6.2.2 Reactive actions

As mentioned in Chapter 2, reactive actions are triggered by action selection rules. These rules are represented by schemas similar to the goal selection rules. The rule for `BecomeDistracted` (shown in Table 6.9) has preconditions that agens is a cat, patiens is a deck and patiens has the slippery attribute.

The selected action is `BecomeDistracted` (shown in Table 6.10), which is quite straightforward.

But even together, these two do not help Scurvy derive that O'Malley will become distracted. Scurvy's planner does not account for the actions of O'Malley and even if it did, this is a reactive action, not a planned one. So to make Scurvy figure out that O'Malley will become distracted, we have to make use of an expectation schema (shown in Table 6.11). This tells the planner that when certain conditions are met, some effect can be expected to follow.

By adding the `ExpectDistracted` schema to an active plan for the distract goal, the planner

Table 6.5: Second goal selection rule for Distract

DistractRule2	
Preconditions	
true	rule(Agens, owlr:typeOrSubType, scs:'Rat') rule(Patiens, owlr:typeOrSubType, scs:'Cat') rule(Patiens, swcr:visibleFor, Agens, ooc) rule(Food, owlr:typeOrSubType, scs:'Food') rule(Food, swcr:visibleFor, Agens, ooc)
false	fact(Patiens, swc:hasAttribute, scs:distracted) fact(Agens, swc:has, Food)
Selected Goal (Goal)	
	goal_schema(Goal) schema_type(Goal, scs:'Distract') schema_agens(Goal, Agens) schema_patiens(Goal,Patiens)

Table 6.6: The HideWithFood goal

HideWithFood	
Preconditions	
true	rule(Agens, owlr:typeOrSubType, scs:'Rat') rule(Patiens, owlr:typeOrSubType, scs:'Food') rule(Target, owlr:typeOrSubType, scs:'Rathole') fact(Agens, swc:has, Patiens)
false	fact(Agens, swc:at, Target)
Success conditions	
true	fact(Agens, swc:has, Patiens) fact(Agens, swc:at, Target)

can derive that a slippery deck will distract O'Malley. Kicking the bucket will make the deck slippery and thus fulfill the precondition for the expectation. The action selection rule makes sure that the expectation comes true by forcing O'Malley to perform the expected behaviour. When performing the `SwabDeck` action, a secondary effect is that O'Malley will no longer be distracted.

We have added a similar construction for the case when Scurvy gets caught while trying to execute a plan to distract O'Malley by dropping a cannonball on his foot. He must already have the cannonball or he will be out of options. By giving Scurvy the expectation that he will be released if he continues his plan to drop the cannonball, we ensure that the planner can still complete the plan. Otherwise, the preconditions for the walk action would fail because Scurvy cannot walk while he is caught, so he would give up trying to complete his plan. An action selection rule makes O'Malley actually perform the action to let Scurvy go when he is distracted.

6.3 Evaluation

Having added these new goals, actions and other schemas, we start up the storyteller and start generating a story. For the first story, Scurvy starts at his lair and O'Malley starts at the galley. The cheese is in the captain's cabin. An image showing the layout of the ship can be found below.

Scurvy immediately takes the goal to find a piece of cheese. Since he does not know where the cheese is, he has to make an assumption about its location. As the story has just started and no conflict has taken place yet, he picks the `AssumeObjectAtWrongLocation`-operator. This makes him believe that the cheese is at O'Malley's location, in the galley, by using OOC knowledge.

Table 6.7: The CatchRat goal

CatchRat	
Preconditions	
true	rule(Agens, owlr:typeOrSubType, scs:'Cat') rule(Patiens, owlr:typeOrSubType, scs:'Rat') rule(Patiens, swcr:visibleFor, Agens, [ooc])
false	fact(Patiens, scs:caughtBy, Agens)
Success conditions	
true	fact(Patiens, scs:caughtBy, Agens)

Table 6.8: The CleanDeck goal

CleanDeck	
Preconditions	
true	rule(Agens, owlr:typeOrSubType, scs:'Cat') rule(Patiens, owlr:typeOrSubType, scs:'Deck') fact(Patiens, swc:hasAttribute, scs:slippery)
Success conditions	
false	fact(Patiens, swc:hasAttribute, scs:slippery)

He proceeds to look there. As soon as Scurvy opens the door, O'Malley discovers him and takes up the goal to catch him. Scurvy in turn tries to run away, but soon finds himself cornered in the Crew's quarters. He then decides he has to distract O'Malley in order to get away. He uses the cannonball to achieve this. As O'Malley is now distracted, Scurvy can get back to the goal of finding the cheese. Since he has discovered that the cheese was not at the galley, he makes a new assumption. This time he can use the `AssumeObjectAtRightLocation`-operator because the conflict has already taken place. This makes him believe the cheese is at the captain's cabin, again by using OOC knowledge. He grabs it and then is given his resolution-goal, to take the cheese back to his lair. In the meantime, O'Malley has recovered from his sore foot, but as Scurvy is constantly moving, he is unable to complete the catch action. Scurvy makes it back to his lair safely and O'Malley can no longer reach him, so the story ends. Below is the output generated for this story by the plot agent (not the narrator). It has been edited for readability and some extra lines have been added to narrate the assumptions and perceptions since the plot agent does not narrate them.

Scurvy wants to get the piece of cheese.
Scurvy assumes the cheese is in the galley.
Scurvy walks to the cargo hold via the door.
Scurvy walks to the gun deck via the ladder.
Scurvy opens the door to the Galley.
Scurvy can see the cheese is not in the galley.
O'Malley sees Scurvy.
O'Malley wants to catch Scurvy.
O'Malley walks to the gun deck via the door.
Scurvy wants to avoid O'Malley.
Scurvy walks to the crew's quarters via the door.
Scurvy wants to distract O'Malley.
Scurvy walks to the gun deck via the door.
O'Malley walks to the crew's quarters via the door.
O'Malley walks to the gun deck via the door.
Scurvy picks up the cannonball from the gun deck.

Table 6.9: Action selection rule for BecomeDistracted

BecomeDistractedRule	
Preconditions	
true	rule(Agens, owlr:typeOrSubType, scs:'Cat') rule(Patiens, owlr:typeOrSubType, scs:'Deck') fact(Patiens, swc:hasAttribute, scs:slippery)
Selected Action (Act)	
	action_schema(Act) schema_type(Act, scs:'BecomeDistracted') schema_agens(Act, Agens) schema_patiens(Act,Patiens)

Table 6.10: The BecomeDistracted action

BecomeDistracted	
Preconditions	
true	rule(Agens, swcr:knowsAction, scs:'BecomeDistracted') rule(Loc, owlr:typeOrSubType, scs:'Deck') fact(Loc, swc:hasAttribute, scs:slippery)
false	fact(Agens, swc:hasAttribute, scs:distracted)
Effects	
true	fact(Agens, swc:hasAttribute, scs:distracted)

Scurvy drops the cannonball on O'Malley's foot.
Scurvy assumes the cheese is in the captain's cabin.
Scurvy walks to the deck via the ladder.
O'Malley dances around on one foot, screaming loudly.
Scurvy opens the door to the captain's cabin.
Scurvy walks to the captain's cabin via the door.
Scurvy picks up the piece of cheese from the captain's cabin.
O'Malley walks to the deck via the ladder.
Scurvy wants to hide with the piece of cheese at Scurvy's lair.
Scurvy walks to the deck via the door.
O'Malley walks to the captain's cabin via the door.
O'Malley walks to the deck via the door.
Scurvy walks to the gun deck via the ladder.
Scurvy walks to the cargo hold via the ladder.
O'Malley walks to the gun deck via the ladder.
Scurvy walks to the brig via the door.
O'Malley walks to the cargo hold via the ladder.
Scurvy crawls to Scurvy's lair via the small hole.
O'Malley walks to the brig via the door.

As we can see in this story, the guided assumptions work as intended. Despite the fact that O'Malley is behind a closed door from where he would normally be unlikely to spot Scurvy, the story is guided in such a way that they do run into each other. This ensures that the conflict takes place and the new operators are used. By making use of actor knowledge, we avoid having Scurvy search in places that do not lead to interesting developments. The result is a reasonable dramatic arc. The hero sets out on an adventure, has to search for his prize, runs into the villain, is chased into a corner, he defeats the villain, finds his prize and finally escapes with the villain breathing down his neck.

Table 6.11: The expectation schema for expecting O'Malley to be distracted

ExpectDistracted	
Preconditions	
true	rule(Agens, owlr:typeOrSubType, scs:'Rat') rule(Patiens, owlr:typeOrSubType, scs:'Cat') rule(Instrument, owlr:typeOrSubType, scs:'Deck') fact(Instrument, swc:hasAttribute, scs:slippery) rule(Instrument, swcr:visibleFor, Patiens, ooc)
Effects	
true	fact(Patiens, swc:hasAttribute, scs:distracted)

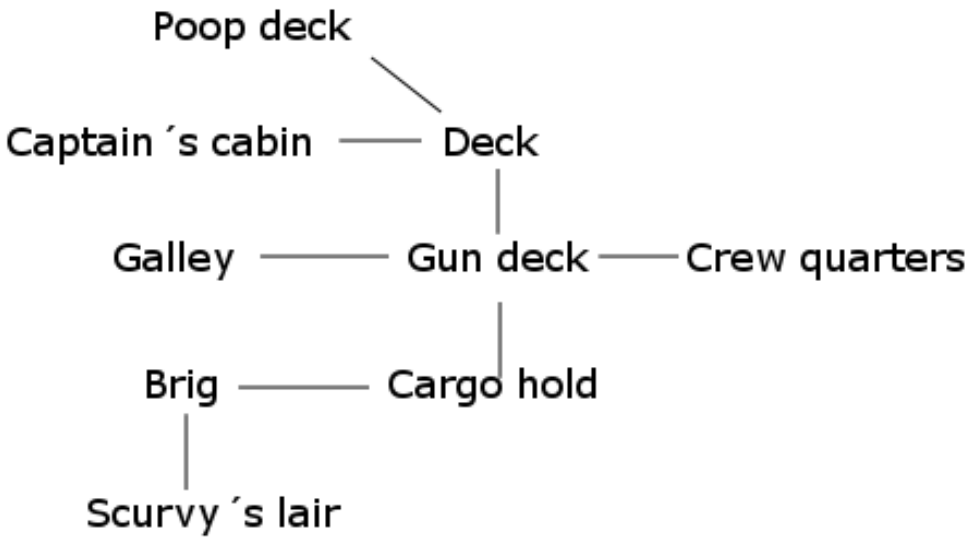

Figure 6.1: Schematic view of the ship layout

After making some slight changes to the storyworld, we start the generation process again. This time, Scurvy begins on the poop deck and O'Malley begins on the gun deck. The cheese is still in the captain's cabin. Once again, Scurvy starts off by making a false assumption. O'Malley starts chasing and Scurvy is forced to retreat to the poop deck. Unable to move further in this direction, he has to think of a distraction again. This time he spots the bucket of soapy water on the deck and decides to kick it over. O'Malley, reluctant to get in trouble with the captain, decides he had better clean this mess up right away, giving Scurvy a chance to go for the cheese. O'Malley finishes his task just as Scurvy returns from the captain's cabin with the cheese. He chases Scurvy again but as before, the quick little rat escapes through his tiny tunnel out of O'Malley's reach. The (edited) transcript of this story can be found below.

Scurvy wants to get the piece of cheese.
Scurvy assumes the cheese is on the gun deck.
Scurvy walks to the deck via the ladder.
Scurvy can see the cheese is not on the gun deck.
Scurvy wants to avoid O'Malley.
Scurvy walks to the poop deck via the ladder.

O'Malley sees Scurvy.
O'Malley wants to catch Scurvy.
O'Malley walks to the deck via the ladder.
Scurvy wants to distract O'Malley.
Scurvy walks to the deck via the ladder.
O'Malley walks to the poop deck via the ladder.
O'Malley walks to the deck via the ladder.
Scurvy kicks over the bucket of soapy water.
O'Malley is distracted by the slippery deck.
O'Malley wants to clean the deck.
Scurvy assumes the cheese is in the captain's cabin.
Scurvy opens the door to the captain's cabin.
O'Malley walks to the poop deck via the ladder.
O'Malley picks up the swab from the poop deck.
Scurvy walks to the captain's cabin via the door.
Scurvy picks up the piece of cheese from the captain's cabin.
O'Malley walks to the deck via the ladder.
Scurvy wants to hide with the piece of cheese at Scurvy's lair.
Scurvy walks to the deck via the door.
O'Malley swabs the deck.
Scurvy walks to the gun deck via the ladder.
Scurvy walks to the cargo hold via the ladder.
O'Malley walks to the gun deck via the ladder.
Scurvy walks to the brig via the door.
O'Malley walks to the cargo hold via the ladder.
Scurvy crawls to Scurvy's lair via the small hole.

The purpose of this second story is to show that a small change in the initial situation can have a big impact on the resulting story. Scurvy's first (false) assumption has little impact on the story because he had little choice but to head to the deck. But as the bucket of soapy water is now located much more conveniently than the cannonball, he uses that in his distraction plan. Since the conflict has now taken place already, the actor knowledge is used to guide him towards the cheese instead of dragging the story on with uninteresting searches in wrong locations. This helps maintain the pace of the story and prevents actions that don't really add anything of value. If the perception system is combined with the interactive storyteller in the future, this may become even more important. Since the interactive version cannot fall back on a narrator to filter out or condense uninteresting parts of the story, it will be preferable to keep things interesting as much as possible.

From the tests in the previous chapter we already concluded that the perception system works as intended. We have made it possible to hide information from characters and give them false beliefs. The stories generated in this chapter illustrate how the split between in-character and out-of-character knowledge can be used. The characters will always try to make use of their IC knowledge and base their plans on that so long as the information is sufficient to make a complete plan. Whenever this is no longer the case because a belief proved to be false, the actor-knowledge is used in combination with past beliefs from the episodic memory. The latter helps the character to make believable choices as they will not search in places they have recently visited. The former is used to direct the character towards interesting places depending on the current state of the plot. The stories also show that characters can be deceived, as in fact the actor component deceives the character and gives it false beliefs.

In both stories we still see that the characters don't take each other into account most of the time. For those cases that we specified, the distracted property works and Scurvy behaves as expected. But in other places, the characters still ignore what the other is doing, which results

in the characters walking past each other. It would have been nicer if O'Malley would wait if he saw that Scurvy was already coming in his direction.

In these stories, the assumption operators have been tailored specifically to this domain. There have been only two locations that the actor could select as being interesting, namely the location of O'Malley or the location of the cheese. In future work, more options should be added and if possible, it should be kept high level. The actor might be given OOC-goals for the character specifically designed to lead to interesting situations. The assumptions should not specifically mention which situations are interesting but instead, the actor should employ assumptions in such a way that they help achieve the OOC-goals while still staying believable according to what the character knows.

One problem with the implementation of the assumption-operators is that they are not yet narrated. This makes it difficult to see when and why a character makes an assumption. In part this is because of the way the filtering of the fabula works. as explained in Chapter 5. Another cause is that internal elements are processed directly by the character unlike most other operators, which are passed through the plot agent and the world agent. The world agent doesn't need to know about internal character processes, but the plot agent should at least be informed if an internal element is processed, so that it can be narrated. This has not yet been added to the implementation and will be left for future work.

Perceptions are also not narrated. This is how it was originally intended because otherwise the stories would describe all the perceptions the beliefs that followed from them. For example, a story might read like this. *Scurvy picks up the piece of cheese from the captain's cabin. Scurvy sees that he has the piece of cheese. Scurvy believes that he has the piece of cheese.* It was decided that this would not make for an interesting read so the perceptions and beliefs were all filtered out of the narration. However, with the new implementation, there are some perceptions that will be important for the reader to know about in order to understand why a character does or doesn't react to a certain event. In the above stories some of these perceptions have been added to make the stories easier to understand. Eventually there should be a mechanism to determine which perceptions are important to narrate. Any perception that reveals a fact that is a precondition for a goal would be a likely candidate to be narrated as would any perception that contradicts a character's beliefs. This is also something that can be added in future work.

6.4 Conclusion

In this chapter we have shown parts of the authoring process for the VST. We added new goals and actions to Scurvy's domain in order to generate a more complete storyline. We employed more of the various types of schemas available for use in the VST. Most notably, we expanded the new assumption operators so that they would guide the story in interesting directions. The stories that were generated show that this method works as intended. The conflict between Scurvy and O'Malley takes place in both cases and afterwards, the story is resolved at a steady pace. A remaining issue is that the characters still don't take each other into account when planning actions and because of that we have to use a workaround to get the desired behaviour. But even then, this only works in the specific situations that the workaround was designed for. In other situations the characters still blindly ignore the actions the other is performing. Another issue that still remains is that the assumption operators are difficult to trace as they are left out of both the fabula output and the narration. Finally, a method is still needed to determine which perceptions are worth narrating.

7 Conclusions & Recommendations

The goal of this project was to improve the characters in the Virtual Storyteller by turning them from omniscient characters into more believable ones whose knowledge of the storyworld is limited by what they could realistically perceive. The ultimate purpose of this endeavour was to enable the generation of new types of stories and to make the generated stories more enjoyable. In this chapter we will look back at the work that has been done to achieve this goal, as well as give some recommendations for future work.

7.1 Conclusions

This research started by examining the workings of the VST closely, as well as taking a look at other similar systems. This gave a better insight into which parts of the VST would be affected by implementing perceptions. In Chapter 1 two research questions were posed.

> Research question 1a: How can perceptions be blocked for characters?
>
> Research question 1b: How can the character be made to deal with missing information?

Since the character agents in the VST perform a dual role as actors as well as characters, the decision was made to divide their knowledge base into separate graphs. To achieve this, we added a fourth argument to the knowledge triples that stores to which knowledge graph the triple belongs. Graphs were divided into in-character knowledge and out-of-character knowledge, the first stores the character's current knowledge of the storyworld and the second stores the actual world state for the actor. Besides those two, a graph was also added for core knowledge for ontology information, the description of objects and concepts that can exist in the storyworld, which is needed by both the actor and the character as well as a fabula graph to store the character's episodic memory, all the actions, goals, perceptions, events et cetera that have happened in the story as far as the character knows. Keeping the actual world information was a requirement for the actor to be able to perform its tasks such as framing operators and goal justification.

The decision was made to let the plot agent send all perceptions to all characters as it did before because the characters require this information on the actor level anyway. The decision which perceptions to pass on to the character level to adopt as beliefs was left to the actor level. In order to make this decision a system of rules was implemented. This answers research question 1a. For this research only the sense of sight was implemented. The rule system can be expanded as the storyworld requires and other senses can be added to it at a later time.

Each character is given his own copy of the files that describe the initial world state so that characters can have false or missing beliefs right from the start of the story. The character agents keep track of false beliefs in order to trigger the correct perceptions when they discover that they were wrong.

In some cases a character will get a perception that invalidates a belief but does not provide the information to fill in the correct belief. For such cases a method was needed for the character to

fill in the missing facts. To do this a new assumption operator was implemented that allows a character to make a believable guess about the missing facts. The guess can be wrong but even then it will be enough to make the character act on it, which will eventually lead to the discovery that the guess was wrong. In the mean time, the story will keep moving and the character may even find out the true facts during his actions. This answers research question 1b.

Since random searching doesn't really add much to the excitement of the story, the choice was made to give a storyworld author some plot control options through the assumption operators. When the character has a decision to make about where to search for information, we have a good opportunity to direct the character towards a more interesting situation. Making use of the actual world knowledge the actor keeps, the character can be directed either to the correct location or to a location that adds some conflict and excitement to the story.

During the evaluation phase these assumptions were used to direct the hero, Scurvy, directly towards the villain, O'Malley, in order to force conflict between the two. After the conflict happened, another assumption guided Scurvy towards his goal in order to resolve the story. During the authoring of the storyworld for these stories, workarounds had to be found for some of the flaws in the behaviour of the character agents. Characters only learn about actions through perceptions and these are only sent when an action is completed. Because of this, characters have no way of anticipating the results of actions currently being performed by others. One of the consequences of this was that characters had no way of knowing when another character was distracted so we had to explicitly tell them about this. This was important to let Scurvy know he could work on other goals because he was no longer in direct danger of being captured.

The evaluation stories show that the perception system and the assumption operators work as intended. The characters now act on beliefs that they could logically have instead of knowing and reacting to every change in the world as it happens. This makes their behaviour more believable. The guided assumptions let the characters make wrong decisions and can be used to force conflicts to happen even if a character would have avoided them if he had known everything.

It is not yet possible to narrate the assumptions and let the audience know about them due to the way internal element operators were initially implemented. This is something that should be fixed for future versions. Regardless, the evaluation stories demonstrate storylines that would not have been possible without the perception system.

7.2 Recommendations for future work

During this project a number of ideas were conceived that could not be realized within the available time. Some of these ideas are directly related to the areas of perception and assumption but there are also some recommendations for the VST in general. This section will mention some of these ideas for future work.

7.2.1 Different senses

For this project only the sense of sight was implemented because the first objective was to limit the amount of perceptions a character would receive. Implementing at least the other traditional senses (hearing, touch, smell and taste) would be good to increase the amount of stories that can be told. Of course the usefulness of each of these senses depends on how they will be used in the story that an author wants to tell. If a story is about pirates finding treasure and shooting cannons, taste and smell might not add much. If however the story is about stolen rum that has been replaced with water, they would become much more important. Hearing in particular will become more important when the characters are given the ability to speak and exchange information. Having other senses would be critical for stories in which a character has one of their senses temporarily or permanently disabled, such as wearing a blindfold or being deaf.

This could lead to interesting stories.

7.2.2 Obviousness

The decision which perceptions a character will actually see is currently based purely on rules. For instance, characters can see everything that happens at their current location. Sometimes there are some effects that may be visible according to the rules but it may be more interesting for the story if the character does not notice them, see section 2.5.3. For example, a character who has his pockets picked would currently always notice this according to the rules, but for the story it might be better if this is only discovered later. For these cases it would be good if there was a method to determine how obvious an effect is to a character, based upon factors like the size of an object, the loudness of a sound and also the objects or task that have the character's attention. The character would notice that his wallet is missing when he tries to take it out of his pocket, while he may not have noticed the theft itself because he was focused on something else.

7.2.3 Certainty

Ideally, there would be a difference in certainty between a belief derived from a perception and one derived from an assumption. But currently the VST has no mechanism to make this distinction, nor does the planner have a way to use such information. If an emotional system is implemented, certainty levels would likely have an impact on plan success chance and the resulting emotional reaction. FearNot! for example uses plan probability to determine the amount of hope and fear generated by a goal. Uncertainty about information would have an impact on plan probability, and thus the resulting emotions.

7.2.4 Narration

Currently the assumptions are not narrated because internal element operators are not sent to the plot agent by the characters. This has to be fixed so that assumptions can be properly narrated. Perceptions are also not narrated by design because most of them are self-evident. Now that characters can no longer perceive everything, some perceptions should be narrated so the audience knows about them. A method should be found to find those perceptions that the audience should be informed of and they should be narrated.

7.2.5 Actor goals

For the evaluation stories, the guided assumptions were specifically tailored to a single goal, to force the conflict between Scurvy and O'Malley by guiding Scurvy to O'Malley's location. Eventually it would be useful to have the option of giving the actor its own goals, separate from the goals of the character. These goals would be more focused on creating an engaging story whereas the character's goals are more focused on creating believable behaviour. If the actor has such goals, the assumption could be implemented in a more general manner. The actor would pick any assumption that contributes toward achieving an actor goal while still remaining believable for the character.

For example, you will rarely see a character in a story go to the bathroom just to use the toilet. They will find a clue, have an important conversation, or the other way around, they miss out on something important because they were out of the room. If nothing of consequence happens then it could just as well be left out of the story. If assumptions can be given direction so that the resulting plan has interesting consequences then the story should be more interesting. To do this, assumptions will need additional preconditions that specify what is interesting. One of the factors that play a role in this is something that is completely out of the realm of character

knowledge, which is the story phase. The character should not be aware that he is part of a story, let alone what phase it is in. Still, story phase has a major influence on the flow of events during a story. During the rising action phase, conflicts will build up and new side conflicts may be started. When the climax has passed, during the dénouement, the open conflicts should be resolved. The characters have no knowledge of these guidelines and will not change their behaviour accordingly. On the one hand this is desirable because we want the characters to behave naturally. On the other hand, the occurance of interesting events is mostly left up to chance. The initial story domain plays an important role in this, but as mentioned above, how interesting an event is depends much on how it fits into the story as it evolves. Adding only interesting events to the storyworld is not an option, because even a mundane action of going to the toilet can have interesting consequences if it happens at the right time and place.

A question is how specific this should be. On the domain level, certain elements of the storyworld could be marked as interesting. For instance, O'Malley would be a source of conflict for Scurvy. If conflict is what is required in the story at that point, the assumption for the location of the cheese could return O'Malley's location instead. More generally, an assumption which fits into a plan that establishes the preconditions of a side conflict would lead to a more interesting story. If in turn the side conflict also contributes to the main conflict, the story will be more coherent. Reaching this level of guidance would require high level planning that can take multiple goals in account, possibly including actor goals which have to be treated as such and may not interfere with character believability. As the number of options increases, the guidance should be kept high level so that the characters still have the freedom to fill in the details according to their own personality. This could make use of motivations similar to those introduced for the characters [Bra10]. Certain conflicts could be marked as likely to lead to violence. Then when the plot requires violence, the actor could use its influence to try and meet the conditions for one of these conflicts. The character then uses his own motivations when choosing how to resolve the conflict. In a storyworld that has a good number of possible conflicts, the number of stories that can be told will increase dramatically and by using actor goals and assumptions, these conflicts are much more likely to be triggered at the desired time in the story.

7.2.6 OOC fabula

The character agent currently only stores fabula that is in-character, see section 4.5. All the actions and events that happened in the story so far that the character has seen, all the goals, beliefs and perceptions that the character has had so far. It would be good for the actor to also have access to fabula that is out of character such as the goals of other characters or information about what another character has seen. When used in combination with actor goals, this may help an actor to choose goals that directly conflict with the goals of another character in order to instigate conflict or goals that indirectly help another character in order to relieve tension, depending on what is required for the story. For the stories that are currently being generated by the VST this would not be necessary as they mostly consist of a single conflict and the characters usually have specific goals assigned to them. But in a larger world where characters have many goals to choose from, it will be more difficult if not impossible to determine which goals are the most suitable for the developing plot without knowing what the other characters' goals are.

7.2.7 Anticipating actions

One problem that became obvious during the authoring of the evaluation stories was the fact that characters do not anticipate what the other characters are doing. In our evaluation stories, O'Malley has the goal to capture Scurvy. O'Malley will keep moving to the location where Scurvy currently is, even if Scurvy is already moving to the location where O'Malley is. They just walk past each other in the doorway, which seems odd. Characters should at least take into account the current actions of characters they can see. They should either include the effects of these actions

into their own plan if they are beneficial, or they should try to prevent the action otherwise. For the doorway example, O'Malley should see that Scurvy is moving towards him and should not plan an action to go to Scurvy's current location since he is already at the location where he expects Scurvy to be. Or he could try to close the door to prevent Scurvy from leaving. Using the perception rules, it is already possible to determine which characters are visible for someone, and consequently which actions are visible. The problem is that the world agent only broadcasts the effects of an action after it has been completed. It should also broadcast the fact that an action has started. The effects of actions performed by other characters that are visible for a character should then be added to their plan as plan steps. The planner's threat resolution should be able to determine whether the effects of such a step threaten another step in a character's plan which would require extra steps to invalidate a precondition of the threatening step, thus preventing the action from being completed. Reasoning about another character's goals and anticipating future actions would be even better but also a lot more difficult to implement.

7.2.8 Final thoughts

The evaluation stories already showed that small changes to a storyworld can have a substantial effect on the generated story. If we can combine actors who have their own goals with guided assumptions and a storyworld that has several conflicts and goals to choose from, the Virtual Storyteller would be able to produce a lot of different stories from a single setting. And because of the actor guidance, the characters would get into interesting situations on a regular basis. If it is also combined with the Interactive Storyteller, users would be able to experience different stories every time they use it. Such a system would make the work that is put into the authoring of a storyworld pay off even more, by generating many more stories out of it.

Appendices

A

Separating knowledge

A.1 RDF load functions

The standard load functions that come with SWI-Prolog's RDF database module do not allow loading more than one file into the same graph. Loading another file directly into a graph with the same name will empty the graph first. To get around this restriction a separate load function had to be written that loads the file into a temporary graph first and then copies it into the target graph. However copying data from one graph to another is also a bit tricky because the database does not allow concurrent transactions which prevents us from copying the triples from the temporary graph straight into the target graph. Instead the temporary graph is copied to a list first, from where the triples can be written into the target graph one at a time. The temporary graph can be emptied and reused for loading the other files.

```
graphToList(InGraph,OutList) :-
    findall([S,P,O], rdf(S,P,O,InGraph:_),OutList).

assertListToGraph([],_OutGraph).
assertListToGraph([[S,P,O]|RestList],OutGraph) :-
    rdf_assert(S,P,O,OutGraph),assertListToGraph(RestList,OutGraph).

copyToGraph(InGraph,OutGraph) :-
    graphToList(InGraph,TempList),
    assertListToGraph(TempList,OutGraph).

rdfLoadToGraph(Url, GraphName) :-
    rdf_db:rdf_load(Url,[if(true),db(tempGraph)]),
    copyToGraph(tempGraph, GraphName),
    rdf_unload(tempGraph).
```

A.2 Incoming knowledge

Using the new load functions, the initial knowledge is placed in the correct graphs. The next step is to make sure that new information is also written to the correct graphs. For the world agent this is very easy. It only needs to keeps its OOC graph up to date as that is where it stores the current world setting. For the plot agent, a few more changes need to be made. Any operator results it gets from the world agent need to be written to its OOC graph only. For positive facts, omitting the graph argument doesn't cause much harm but for negative facts, this will cause it to remove the fact from all graphs, including any fabula element that contains this fact. If something changes the current world state, that doesn't mean it should also alter past events.

Incoming fabula elements will be written to subgraphs named after the unique individual they describe. As mentioned before this was already done, but the assertions have been redirected to the RDF database. This means that queries to fabula elements now pass through the OWL reasoning module. Apart from that, fabula is now saved as well when the RDF database is saved.

Causalities and descriptions of the fabula elements are written to a graph called fabula, whereas before they were in the main graph.

For the character agent, things get more interesting. It will at least need to write all new world knowledge to its OOC graph. For the IC knowledge we have to apply the perception rules that will determine what will be copied to the IC graph.

A.3 Visibility rules

These are the rules used to determine what a character can see.

```
% Determine if an object is located at the specified location in some way

% Directly
rule(O, 'http://www.owl-ontologies.com/SWCRules.owl#objectAtLocation', L,
    G):-
  query(O, 'http://www.owl-ontologies.com/StoryWorldCore.owl#at', L, G).

% If an object is held by a character, it is at that character's location
rule(O, 'http://www.owl-ontologies.com/SWCRules.owl#objectAtLocation', L,
    G):-
  query(C, 'http://www.owl-ontologies.com/StoryWorldCore.owl#has', O, G),
  rule(C, 'http://www.owl-ontologies.com/SWCRules.owl#objectAtLocation', L
      , G).

% Similar to being held
rule(O, 'http://www.owl-ontologies.com/SWCRules.owl#objectAtLocation', L,
    G):-
  query(O, 'http://www.owl-ontologies.com/StoryWorldSettings/Scurvy.owl#
      caughtBy', C, G),
  rule(C, 'http://www.owl-ontologies.com/SWCRules.owl#objectAtLocation', L
      , G).

% Determines if an object is visible for a character.
rule(O, 'http://www.owl-ontologies.com/SWCRules.owl#visibleFor', C, G):-
  rule(O,   'http://www.owl-ontologies.com/SWCRules.owl#visibleFrom', L, G
      ),
  rule(C,   'http://www.owl-ontologies.com/SWCRules.owl#objectAtLocation',
       L, G),
  unpQuery(fact(O, 'http://www.owl-ontologies.com/StoryWorldCore.owl#
      hasAttribute', 'http://www.owl-ontologies.com/StoryWorldCore.owl#
      hidden',G)),!.
```

```
% Determines if an object is visible from a location.

% Same location
rule(L1, 'http://www.owl-ontologies.com/SWCRules.owl#visibleFrom', L1, _G)
    .

% Object at L1 visible from L2 if L1 is visible from L2
rule(O, 'http://www.owl-ontologies.com/SWCRules.owl#visibleFrom', L1, G):-
  rule(O, 'http://www.owl-ontologies.com/SWCRules.owl#objectAtLocation',
      L2, G),
  rule(L2, 'http://www.owl-ontologies.com/SWCRules.owl#visibleFrom', L1, G
      ).

% Location L2 is visible from L1 if there is a path from L1 to L2 and in
    case there is a door, it is open
rule(L2, 'http://www.owl-ontologies.com/SWCRules.owl#visibleFrom', L1, G)
    :-
  query(rule(L1, 'http://www.owl-ontologies.com/OWLRules.owl#isNot', L2)),
  query(rule(P,   owlr:typeOrSubType, 'http://www.owl-ontologies.com/
      StoryWorldSettings/Scurvy.owl#NormalPath')),
  query(P,  'http://www.owl-ontologies.com/StoryWorldCore.owl#from',  L1,
      G),
  query(P,  'http://www.owl-ontologies.com/StoryWorldCore.owl#to',  L2, G)
      ,
  query(P,  'http://www.owl-ontologies.com/StoryWorldCore.owl#hasDoor', D,
       G),
  query(D,  'http://www.owl-ontologies.com/StoryWorldCore.owl#hasAttribute
      ',  'http://www.owl-ontologies.com/StoryWorldCore.owl#open', G).

% Case for no door
rule(L2, 'http://www.owl-ontologies.com/SWCRules.owl#visibleFrom', L1, G)
    :-
  query(rule(L1, 'http://www.owl-ontologies.com/OWLRules.owl#isNot', L2)),
  query(rule(P,   owlr:typeOrSubType, 'http://www.owl-ontologies.com/
      StoryWorldSettings/Scurvy.owl#NormalPath')),
  query(P,  'http://www.owl-ontologies.com/StoryWorldCore.owl#from',  L1,
      G),
  query(P,  'http://www.owl-ontologies.com/StoryWorldCore.owl#to',  L2, G)
      ,
  unpQuery(fact(P,  'http://www.owl-ontologies.com/StoryWorldCore.owl#
      hasDoor', _D, G)).

% Objects in open containers are visible from the container's location.
rule(O, 'http://www.owl-ontologies.com/SWCRules.owl#visibleFrom', L, G):-
  rule(B, 'http://www.owl-ontologies.com/SWCRules.owl#objectAtLocation', L
      , G),
  query(B, 'http://www.owl-ontologies.com/StoryWorldCore.owl#contains', O,
       G),
  query(B,  'http://www.owl-ontologies.com/StoryWorldCore.owl#hasAttribute
      ',  'http://www.owl-ontologies.com/StoryWorldCore.owl#open', G).

% Special case for door and the like (related to path, not AT a location)
% Objects on a path are visible from location if path is from location
rule(O, 'http://www.owl-ontologies.com/SWCRules.owl#visibleFrom', L, G):-
  query(P, 'http://www.owl-ontologies.com/StoryWorldCore.owl#from', L, G),
  query(P, 'http://www.owl-ontologies.com/StoryWorldCore.owl#hasDoor', O,
      G).
```

B

Debugging tools

In order to monitor whether all the perceptions were being added properly and at the right time, some new functions were implemented. The first allows to save the entire knowledge base to a readable textfile. This helped greatly in discovering a lot of triples that were being stored multiple times. For instance, the entire core data was being read in twice which previously remained undetected. Similarly, perceptions were being translated to beliefs, and the contained contentfabula was being recursively processed multiple times. These problems have been resolved.

The other new function allows typing a query in the command box of an agent and running the test knowledge base command from the menu to print all the answers to the query. This gives some direct access to the knowledge base and should be used with care. Some exceptions will be caught but others will still cause the agent to crash irrevocably. This does however allow to quickly print out a character's IC graph for instance or query a precondition. Namespaces in the query are not automatically expanded which makes typing the queries correctly a bit of a chore. For myself, I kept queries I used in a separate textfile, along with a saved knowledge base in which all the expanded namespaces are listed. Now it is possible at any moment to test for instance the following query:

```
query(rule(0, 'http://www.owl-ontologies.com/SWCRules.owl#visibleFor',
'http://www.owl-ontologies.com/StoryWorldSettings/Scurvy.owl#Omalley', ooc
    ))
```

As a result, all objects currently visible for O'Malley according to the OOC graph are printed, as can be seen in Figure B.1. These tools can be used to help troubleshoot why characters are not reacting as intended. They can also help when authoring new conditions, by allowing to print the results of the condition at any time during the simulation.

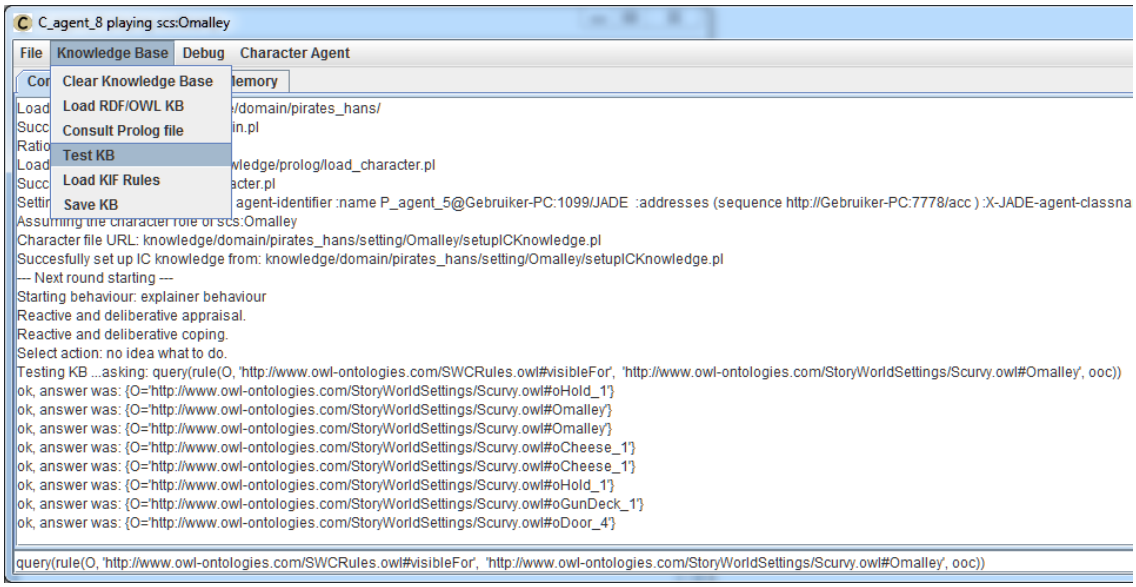

Figure B.1: Result of the test knowledge base function

C

The storyworld

```
################################################
# schematic map of the ship:
#
#
#              poopdeck
#                      \
#    captains cabin - deck
#                         \
#      galley    -    gun deck   -  crew cabin
#                           \
#                  brig - hold
#                 /
#   Scurvy's Lair
#
################################################

# The deck
:oDeck_1
    a   scs:Deck ;
    rdfs:label "the deck" ;
    .

# The poop deck
:oPoopDeck_1
    a   scs:Deck ;
    rdfs:label "the poop deck" ;
    .

# The captain's cabin
:oCapCabin_1
  a   scs:Cabin ;
  rdfs:label "the captain's cabin" ;
  .

# The gun deck
:oGunDeck_1
  a   scs:Cabin ;
  rdfs:label "the gun deck" ;
  .
```

```
# The Galley
:oGalley_1
  a scs:Galley ;
  rdfs:label "the galley" ;
  .

# The crew quarters
:oCrewCabin_1
  a scs:Cabin ;
  rdfs:label "the crew's quarters" ;
  .

# The brig
:oBrig_1
    a   scs:Cabin ;
    rdfs:label "the brig" ;
    .

# Scurvy's Lair
:oLair_1
    a   scs:Rathole ;
    rdfs:label "Scurvy's lair" ;
    .

# The main hold
:oHold_1
    a   scs:Hold ;
    rdfs:label "the cargo hold" ;
    .

#Connecting the parts of the ship

# The ladder from the deck to the gun deck
:oLadder_1
    a   scs:Ladder ;
    swc:from :oDeck_1 ;
    swc:to :oGunDeck_1 ;
    swc:from :oGunDeck_1 ;
    swc:to :oDeck_1 ;
    swc:hasDoor :oHatch_1 ;
    .

  # A closed hatch connecting the deck with the gun deck
  :oHatch_1
      a   swc:Door ;
      swc:hasAttribute swc:openable ;
      swc:hasAttribute swc:open ;
      rdfs:label "the hatch to the gun deck" ;
      .
```

```
# The ladder from the gun deck to the hold
:oLadder_2
    a   scs:Ladder ;
    swc:from :oGunDeck_1 ;
    swc:to :oHold_1 ;
    swc:from :oHold_1 ;
    swc:to :oGunDeck_1 ;
    .

# The ladder from the main deck to the poop deck
:oLadder_3
    a   scs:Ladder ;
    swc:from :oPoopDeck_1 ;
    swc:to :oDeck_1 ;
    swc:from :oDeck_1 ;
    swc:to :oPoopDeck_1 ;
    .

# the path from the main deck to the captain's cabin, (with a lockable
    door)
:oPath_1
    a   scs:NormalPath ;
    swc:from :oDeck_1 ;
    swc:to :oCapCabin_1 ;
    swc:from :oCapCabin_1 ;
    swc:to :oDeck_1 ;
    swc:hasDoor :oDoor_1 ;
    .

  # A closed door connecting the main deck with the captain's cabin
  :oDoor_1
      a   swc:Door ;
      swc:hasAttribute swc:openable ;
      swc:hasAttribute swc:closed ;
      rdfs:label "the door to the captain's cabin" ;
      .

# the path from the gun deck to the galley
:oPath_2
    a   scs:NormalPath ;
    swc:from :oGunDeck_1 ;
    swc:to :oGalley_1 ;
    swc:from :oGalley_1 ;
    swc:to :oGunDeck_1 ;
    swc:hasDoor :oDoor_2 ;
    .

  # An open door connecting the gun deck with the galley
  :oDoor_2
      a   swc:Door ;
      swc:hasAttribute swc:openable ;
      swc:hasAttribute swc:closed ;
      rdfs:label "the door to the Galley" ;
      .
```

```
# the path from the gun deck to the crew's quarters
  :oPath_3
    a   scs:NormalPath ;
    swc:from :oGunDeck_1 ;
    swc:to :oCrewCabin_1 ;
    swc:from :oCrewCabin_1 ;
    swc:to :oGunDeck_1 ;
    swc:hasDoor :oDoor_3 ;
    .

    # An open door connecting the gun deck with the crew quarters
  :oDoor_3
    a   swc:Door ;
    swc:hasAttribute swc:openable ;
    swc:hasAttribute swc:open ;
    rdfs:label "the door to the crew quarters " ;
    .

# the path from the main hold to the brig
:oPath_4
    a   scs:NormalPath ;
    swc:from :oHold_1 ;
    swc:to :oBrig_1 ;
    swc:from :oBrig_1 ;
    swc:to :oHold_1 ;
    swc:hasDoor :oDoor_4 ;
    .

  # A closed door connecting the cargo hold with the brig
  :oDoor_4
      a   swc:Door ;
      swc:hasAttribute swc:openable ;
      swc:hasAttribute swc:open ;
      rdfs:label "the door to the brig" ;
      .

# A tiny passage from the brig to Scurvy's hidden lair
:oRatTunnel_1
  a scs:RatTunnel ;
    swc:from :oBrig_1 ;
    swc:to :oLair_1 ;
    swc:from :oLair_1 ;
    swc:to :oBrig_1 ;
    .
```

```
# Objects aboard the ship.
# Author: Hans ten Brinke
# Date:   1 september 2009
####

# A container
:oTrunk_1
  a swc:Container ;
  swc:at :oCapCabin_1 ;
  swc:hasAttribute swc:openable ;
    swc:hasAttribute swc:closed ;
  rdfs:label "The captains trunk"
  .

# A piece of cheese
:oCheese_1
    a   scs:Food ;
   swc:at :oCapCabin_1 ;
    rdfs:label "the piece of cheese" ;
    .

# A cannonball on the gun deck
:oCannonball_1
    a   scs:Cannonball ;
    swc:at :oGunDeck_1 ;
    rdfs:label "the cannonball" ;
    .

# Soapy water in the bucket
:oSoapyWater_1
  a scs:SoapyWater ;
  rdfs:label "soapy water" ;
  .

# A bucket of soapy water on the poopdeck
:oBucket_1
  a scs:Bucket ;
# swc:at  :oPoopDeck_1 ;
  swc:at  :oDeck_1 ;
  swc:contains  :oSoapyWater_1 ;
  rdfs:label "the bucket" ;
  .

# A swab on the poopdeck
:oSwab_1
  a scs:Swab ;
  swc:at  :oPoopDeck_1 ;
  rdfs:label "the swab" ;
  .
```

Bibliography

[ADP06] Ruth Aylett, Joao Dias, and Ana Paiva. An affectively driven planner for synthetic characters. In *In International Conference on Automated Planning and Scheduling (ICAPS*, pages 2–10, 2006.

[Alo12] T. Alofs. The interactive storyteller: A multi-user tabletop board game interface to support social interaction in ai-based interactive storytelling. Master's thesis, University of Twente, Department of Electrical Engineering, Mathematics and Computer Science, 2012.

[Ari07] Aristotle. *Poetics*. 1907. Trans. by S.H. Butcher.

[Aut] Various Authors. Virtual storyteller wiki. `https://www.dropbox.com/home/Storytelling-shared/VirtualStorytellerwikibestanden`. Community site for the Virtual Storyteller.

[BLR92] Joseph Bates, A. Bryan Loyall, and W. Scott Reilly. An architecture for action, emotion, and social behavior. In *MAAMAW '92: Selected papers from the 4th European Workshop on on Modelling Autonomous Agents in a Multi-Agent World, Artificial Social Systems*, pages 55–68, 1992.

[Bra10] J. Bragt. Towards believable characters in the virtual storyteller. Master's thesis, University of Twente, Department of Electrical Engineering, Mathematics and Computer Science, 2010.

[Cam49] J. Campbell. *The Hero with a thousand faces*. Princeton University Press, 1949.

[CCM02] M Cavazza, F Charles, and S.J. Mead. Character-based interactive storytelling. *IEEE Intelligent Systems, Volume 17, Issue 4*, pages 17–24, 2002.

[Faa02] S. Faas. Virtual storyteller: An approach to computational story telling. Master's thesis, University of Twente, Department of Electrical Engineering, Mathematics and Computer Science, 2002.

[FN72] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3), 1972.

[Fre00] G. Freytag. *Technique of the drama - An exposition of dramatic composition and art*. Scott, Foresman and Company, 1900.

[GvL02] Andrew Gordon and Mike van Lent. Virtual humans as participants vs. virtual humans as actors. In *Proceedings of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, 2002.

[jad] Jade - java agent development framework (version 3.4). `http://jade.tilab.com/index.html`.

[Kru07] E.E. Kruizinga. Planning for character agents in automated storytelling. Master's thesis, University of Twente, Department of Electrical Engineering, Mathematics and Computer Science, 2007.

[Lid04] Lars Lidén. Artificial stupidity: The art of intentional mistakes. In *AI Game Programming Wisdom 2*, pages 41–48. Charles River Media Inc., 2004.

[Mee77] James R. Meehan. Tale-spin, an interactive program that writes stories. In *IJCAI'77: Proceedings of the 5th international joint conference on Artificial intelligence*, pages 91–98, 1977.

[OCC88] Andrew Ortony, Gerald Clore, and Alan Collins. *The cognitive structure of emotions*. Cambridge University Press, 1988.

[owl] W3c web ontology language.

[Pro68] V Propp. *Morphology of the Folktale*. University of Texas Press, 1968.

[Sla06] N. Slabbers. Narration for virtual storytelling. Master's thesis, University of Twente, Department of Electrical Engineering, Mathematics and Computer Science, 2006.

[ST08] I. Swartjes and M. Theune. The virtual storyteller: Story generation by simulation. In *Proceedings of the 20th Belgian-Netherlands Conference on Artificial Intelligence (BNAIC 2008)*, page 257, 2008.

[ST09] I. Swartjes and M. Theune. Late commitment - virtual story characters that can frame their world. Technical Report TR-CTIT-09-18, Human Media Interaction group, Twente University, 2009.

[Swa06] I. Swartjes. The plot thickens: bringing structure and meaning into automated story generation. Master's thesis, University of Twente, Department of Electrical Engineering, Mathematics and Computer Science, 2006.

[Swa10] I. Swartjes. *Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative*. PhD thesis, University of Twente, Department of Electrical Engineering, Mathematics and Computer Science, 2010.

[tB14] Hans ten Brinke. Fatima. Research Topics assignment report, 2014.

[Tom09] P. Tomassen. Wacky pirate hijinks - causing and resolving conflict between autonomous agents in a virtual storyworld. Master's thesis, University of Twente, Department of Electrical Engineering, Mathematics and Computer Science, 2009.

[TROdAH04] M. Theune, S. Rensen, R. Op den Akker, and A. Heylen, D. Nijholt. Emotional characters for automatic plot creation. In *Proceedings of TIDSE 2004*, pages 95–100, 2004.

[tvt] Television tropes and idioms. `http://tvtropes.org/pmwiki/pmwiki.php/Main/HomePage`. This wiki is a catalog of the tricks of the trade for writing fiction.

[Wes09] Mick West. Intelligent mistakes: How to incorporate stupidity into your ai code. `http://www.gamasutra.com/view/feature/3947/intelligent_mistakes_how_to_.php`, 2009.

[Wie] Jan Wielemaker. Swi-prolog. `http://www.swi-prolog.org/`.

[Zee10] R. Zeeders. Comics - comic generation from story content graphs. Master's thesis, University of Twente, Department of Electrical Engineering, Mathematics and Computer Science, 2010.

List of Figures

List of Tables