Text to Adventure Game: Automated Generation of Adventure Games Based on Fiction Text

Submitted in partial fulfilment
of the requirements of the degree
Bachelor of Science (Honours)
of Rhodes University

Ross Berkland

09th November 2009

Abstract

Recent advancements in the Text-to-Scene field of research have lead to the development of a system which automatically extracts key concepts from the text of a fiction book and generates a computer animated movie depicting the story. This project extends this idea by creating a Text-to-Game system capable of converting text into a fully playable game. In order to achieve this we focus on recreating the environment, characters and events described in the text. Evaluation of the system occurs through user tests in which participants are asked to measure the accuracy with which the game represents the events, characters and goals described in the story. It is shown that the game created by the system it an accurate representation of the text. Also, we discuss the system as a rapid prototyping tool and a new method for automated game generation.

Acknowledgments

I would like to thank my supervisor Shaun Bangay for his guidance and constant enthusiasm; The VRSIG for their support and insight; and the Computer Science Department of Rhodes University for provided the facilities which have made this research possible. I also acknowledge the financial and technical support of this project of Telkom SA, Comverse SA, Stortech, Tellabs, Amatole, Mars Technologies, Bright Ideas Projects 39 and THRIP through the Telkom Centre of Excellence at Rhodes University

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Text-to-scene (TTS) systems have become an increasingly popular means for visualising the textual description of an event or story. We extend this by creating an interactive form of TTS system to emulate the experience of adventure books in a virtual world. To achieve this we show that it is possible to convert a book into a game by solving the problems involved in building such a system. This involves translating the environment described in the text into a virtual map; creating virtual characters which accurately match their textual counterparts; and finally by recreating the events of the story as achievable game goals.

While converting a book into a game is one of the primary goals of this research, we also evaluate the system as a new, automated attempt to game development. We aim to show that our Text-to-game system can simplify the game development process by separating the game description from the actual implemented software (source code) giving us a rapid prototyping system for games and a method for customising the final game with ease and efficiency.

1.2 Thesis Outline

The rest of this paper describes related work in various other fields and provides detail into the workings of our system. It adopts the following structure:

- In Chapter 2 we describe some of the research done in related fields and show how it ties into this project. In particular we look at research from the text-to-scene and automated game generation fields.

- Chapter 3 presents the design of the system. In particular we consider what is required to generate the characters, map and events given the textual annotations..
- Chapter describes how the system is implemented. More specifically, we look at the implementation of the various system components.
- Chapter 5 describes an evaluation of the system. First, the experimental methodology is described and then results are discussed.
- In Chapter 5 we provide a summary of our results which is followed by the final conclusion in which we summarise what this research has achieved

Chapter 2

Related Work

2.1 Introduction

Research on text-to-scene systems has become increasingly widespread. The capabilities of these systems have evolved from the ability to find several pictures relating to a piece of text to the generation of fully animated 3-dimensional scenes. Researchers are also discovering a multitude of applications for such systems extending out as far as the legal and law enforcement fields. This is partly due to the rapid growth of digital media as a means for communication and entertainment. Another fast growing area of computer science is that of computer games. As is discussed in later sections of the paper, game development is becoming increasingly popular as not only an educational tool but also a career choice.

This chapter discusses key elements from both the Text-to-Scene and computer game fields and combines them to identify potential components for use in the development of a Text-to-Adventure game system. This is done by examining 3 main areas of research. The first deals with the applications which create a relationship between natural language and digital media such as Parts-of-Speech taggers and Text-to-Scene systems. Secondly, the paper discusses the structures that computer games and their narratives typically tend to assume. Finally, various aspects of game development and design are discussed. While these are significantly different areas of research, we join them in a logical and meaningful manner to define a structure for the Text-to-Game system.

2.2 Natural Language and Digital Media

Digital media such as computer generated movies, digital art and video games have become increasingly popular. However, modeling a 3D scene on a computer is generally more time consuming than describing that same scene in words. This has led to the development of a new field in computer science known as text-to-scene conversion. By taking natural language as input, these systems generate a digital picture or scene with the aim of depicting the scene as accurately as possible. Understanding these systems requires basic knowledge about how they function which is described below. This section also presents details on existing text-to-scene systems and various applications of the concepts behind such systems.

2.2.1 Structure of Natural Language

Text-to-Scene systems must have the ability to find key phrases in the input text to generate the correct contents in the associated scene. This can be done with Parts of Speech (POS) taggers. These applications make use of information about a particular language in order to find those parts of speech which may be relevant. Development of Parts of Speech taggers has been active for many years and papers on the subject can be found dating back to 1980 as can be seen in Svartvik [26]. The basic concept behind these taggers is that of annotation of text. The tagger will parse the given text and on discovering significant words, will tag them with some form of markup language, typically XML Glass and Bangay [10]. These annotations are important in development of the scene-to-adventure game system as will be seen later. As mentioned above and confirmed by Glass and Bangay [10], the interpretation of written text is necessary for the creation of a text-to-scene system. As a first step towards creating an autonomous text-to-scene system, Glass and Bangay [10] offer an evaluation of several of the freely available parts-of-speech taggers. The research also involves combining existing taggers for increased accuracy. However, accuracy of these taggers are not within the scope of this paper so much as the mere fact that these systems exist. The idea of annotating text for the purposes of a text-to-scene system is extended by Glass and Bangay [11] in which they offer a method for speaker identification in fiction books with approximately 80% accuracy. The approach taken in developing the system involves making use of a collection of parts-of-speech taggers (as mentioned above) for the purposes of giving clues for speaker identification. This system could be largely helpful in the development of a scene-to-adventure game system due to the large quantity of narration and dialog in typical fiction books.

2.2.2 Text-to-Scene Systems

2.2.2.1 Existing Text-to-Scene Systems

Many text-to-scene systems have been developed to date, most of which differ in their methodology and output. Typically a text-to-scene system can be described as one which takes text as its input and generates a scene depicting that text. Those systems which adhere to this definition can be placed in one of three categories regarding their output. That is to say that the current text-to-scene systems typically output either pictures (text-to-picture)Zhu et al. [30], still 3D scenes Coyne and Sproat [6], Sproat [25] or animated 3D scenes Ma [18], Glass and Bangay [13]. These systems make use of the above mentioned parts-of-speech taggers in order to annotate the text in meaningful way with some markup language.

Text-to-Picture

A system is introduced by Zhu et al. [30] in which they attempt to augment communication through use of a text-to-picture system. The system functions by first selecting those key-phrases from the given text which are most *picturable*[1]. It then attempts to find pictures which best match the key-phrases by using words similar to the key phrase to search a database of pictures Zhu et al. [30]. These pictures are then positioned based on the text. For example, if the text specified that mug was on a table, the picture of the mug would be placed above the picture of the table.

While the system specified in Zhu et al. [30] is designed for augmented communication, its simplicity may have prevented it from reaching its goal. Using nothing but pictures to represent the idea described by a piece of text may remove and possibly distort the meaning of the text. This is where the true value of modern text-to-scene systems can be seen which is explained next

Static Text-to-Scene

Systems such as Wordseye, detailed by Coyne and Sproat [6], improve on the concept of the above mentioned text-to-picture systems by rendering 3D scenes. This gives systems like Wordseye the ability to not only illustrate the objects mentioned in

[1]Ease with which an idea can be illustrated with pictures

text but also the spatial relation between them. This allows for a clearer and more accurate representation of the scene described by the text. Another important aspect of text-to-scene used in the Wordseye system is the use of 3D models as opposed to the pictures used by Zhu et al. [30].

The Wordseye system makes use of an object database which contains all of the potential models to be used in the rendered scenes Coyne and Sproat [6]. A database such as this one would be crucial to the implementation of a scene-to-game system if the process is to be fully automated. Additional functionality found in the Wordseye system is that of the inferred environment, an extension developed by Sproat [25]. By considering the context of the scene described in the text Sproat [25] is able to infer information about the environment. A text-to-game system would require such functionality as large 3D open worlds are becoming more and more common as the backdrops for adventure games. However, it must be noted that the input used by many of the text-to-scene systems such as Wordseye takes in a specific storytelling structure dissimilar to natural-language text.

Animated Text-to-Scene

The third type of text-to-scene system is those which generate animated 3D scenes. The complexity and potential value of these systems is far greater than that of the simpler systems. Not only do these systems consider the spatial properties of objects but also their temporal nature. It is for this reason that these systems are highly applicable when considering the design of a scene-to-game system. Recently, Glass and Bangay [13] developed a system in which fiction text is converted into a 3D animated scene. The system works by annotating the text through use of the parts-of-speech taggers used in their earlier work Glass and Bangay [10]. An examples of annotated text can be seen in Figure 2.1. The annotated text is then used to create constraints which are solved to create trajectories and spatial relations for the objects Glass and Bangay [12, 13]. Finally, this information is used in creating an animated scene.

While several text-to-scene systems do exist, it is those which have the ability to recognize natural-language text which are of interest in the current context. The CarSim Johansson et al. [14] and WordsEye Coyne and Sproat [6] systems require text which is structured in a specific story form. Systems such as that created by Glass and Bangay [13] and the CONFUCIUS Ma [18] system can use input text extracted directly from unmodified fiction books. This type of functionality would be a great asset to a scene-to-adventure game system in that there are countless fiction texts available with the potential to become fully-fledged adventure games. It should be noted that the constraints used by Glass and Bangay [13] may be unnecessary in a scene-to-game system as we wish to grant the player the freedom to perform their own actions. However, The concept of player freedom and choice will be discussed further in Section 2.3.

<avatar>Anne</avatar> didn't very much like a big brown **<object>cow</object>** who **<transition type="INSIDE" subject="cow">came</transition>** up **<relation type="near" subject="cow" object="her">close<relation>** and stared at her, but it **<transition type="OUTSIDE" subject="it">went</transition>** away when **<avatar>Daddy</avatar>** told it to.

Figure 2.1: Example of text annotated with XML [9]

2.2.2.2 Common components

In designing a Text-to-Game system it is useful to consider the design of these Text-to-Scene systems such as that presented by Glass [9] and Akerberg et al. [2]. An investigation into these systems shows that most stories or texts are made uniquely identifiable by 3 main elements, the environment, characters and events. For example, consider the system offered by Glass [9] which makes use of the annotations shown in 2.1. An investigation of these annotations shows that the system defines characters as being avatars, the environment as being a setting and the events as transitions and relations between characters and objects. Similarly, in the CarSim system offered by Akerberg et al. [2], the vehicles relate to the characters or main actors while the accident can be considered as the event and the road the environment.

2.2.2.3 Other Applications of the Text-to-Scene Concept

The systems mentioned above have valuable applications in various computer science fields. The technology found in such text-to-scene systems has been utilized to great effect elsewhere as well. The CarSim system Johansson et al. [14] was developed for visualizing traffic accidents through a text based description of the event. This technology could prove to be useful in a variety of fields such as law, insurance and safety. Durupinar et al. [7] have created a system for the reconstruction of crime scene photographs. This makes use of text describing the contents of the photograph to create a 3-dimensional reconstruction of the scene.

While these applications may not be applicable to the development of a scene-to-adventure game system, it is interesting to note that the text-to-scene concept is very much in use and supported.

As we can see from the above discussions, the technology which exists in text-to-scene research has progressed significantly. The ability to render 3D scenes found in current text-to-scene systems can be adapted to allow for the creation of computer games. The next section describes how various narrative structures can be used to create more realistic and enjoyable games.

2.3 Narrative and Game Structures

While text-to-scene systems have many components which could be useful in the development of a scene-to-game system, the game design and narrative are also vital parts of creating a game. While the game narrative will obviously be well guided by the fiction text, the interactive nature of the game should allow for the narrative to evolve as the game progresses. This section details the various narrative structures used in interactive media along with the different structures which video games typically follow.

2.3.1 Interactive Narratives

Interactive narratives are those which have multiple story paths. A game which makes use of interactive narratives allows users to make decisions which alter how the story progresses Riedl et al. [22] .

The idea of interactive narratives in games is a very popular concept as increased opportunity for choice lends itself to greater realism. The reason being that human beings are not constrained by those elements which tend to limit a players choices in games. For example, a predefined story or small map which depicts the entire world hinders the amount of freedom which a player has. However, while choice is desirable to a some extent, the lack of a narrative would greatly subtract from the quality of an adventure game. It is for this reason that a trade-off must be made between interactivity and narrative structure. Riedl et al. [22] present this problem as being storytelling versus simulation or a trade-off between control and coherence. In this approach storytelling describes the predefined coherent narrative and simulation is the players ability to interact with the game and make decisions or their control. Riedl et al. [22] approach this problem through the creation of an automated story director which uses a high-level plot outline to guide the player. Louchart et al. [17] take a similar approach to the issue by offering guidelines for narrative authoring such as considering different actions which can be performed by the player and the repercussions thereof.

The idea of interactive narratives can be tied to that of the text-to-scene concept by looking at the Scene-Driver system Wolff et al. [29]. The system, developed by Wolff et al. [29] in 2004 makes use of scenes from an existing children's television show called Tiny Planets. It is represented in the form of a game in which children are required to select dominoes which depict what will happen next in the story. This is another area where the text-to-scene system could potentially be used. As an alternative to re-using the Tiny Planets television show, existing "choose you own adventure" style stories could be used as the narrative while a text-to-scene system could be used in rendering the scene.

The terms interactive narrative and emergent narrative are often used interchangeably to describe those narratives giving the player choice and having alternate paths. However, we must distinguish between those narratives which have predefined alternate paths and those which have alternate paths as gateways to undefined story elements. These emergent narratives will be described in the next section.

2.3.2 Emergent Interactive Narratives

The concept of interactive narratives can be extended to that of emergent narratives which use less definitive story lines. Similarly to interactive narratives users are given the ability to make choices. However, in an emergent narrative those choices which the users make can alter the narrative in new undefined ways. Unlike basic interactive narratives which tend to coerce the player into conforming to some or other storyline, emergent narratives attempt to create new story-lines to accommodate the players decisions. Louchart et al. [16] define the author's role in an emergent narrative and subsequently give guidelines for fulfilling the role at design time.

Similarly to interactive narratives, emergent narratives can be linked to text-to-scene systems. Implementation of a scene-to-adventure game system would be the perfect case of this. The fiction text will be used in constructing a game world and emergent narrative in which the player will be able to explore alternative storylines not mentioned in the text.

2.3.3 Game Structures

While narratives play an important role in creating a game with playable value, the game mechanics must also be considered. These can often work hand in hand with the narratives. Lindley [15] provides several ideas relating to the structure of games in terms of their goals and narratives. While the paper does not offer much in the way of research methodology, some interesting ideas are mentioned by relating to the structure of games. Game structure is highly dependent on the type of narrative used but the way in which the game world could be represented at any given point is also important. For example, the game could follow a tree-like structure whereby each leaf or node would represent some event or scenario Lindley [15]. Alternatively, the game could be stateful whereby attributes would represent all of the objects in the game world and the values of those attributes would determine the current state.

Knowledge of game and narrative structures is important in developing a text-to-game system as the game is dependent on the story. However, in order to use such concepts we must first discover any limitations which implementation may present. The following section discusses the various methods available for game development and how they can be used in the creation of a text-to-game system.

2.4 Game Development

With user created content becoming increasingly popular, more and more tools for game development have emerged. Numerous open source engines and game development tools have been made available. These ranging from entirely new languages to applications purpose built for the development of game creation. This section details the most relevant of these tools and their uses.

2.4.1 Manual Game Development

As mentioned above, a large jump in the availability of game development tools has caused creating games to become increasingly popular. While projects are being undertaken relating to automatic game development, these systems would not be possible without some initial manual development.

2.4.1.1 Programming Based Development

While many of the game development tools currently available do not require any programming language, game development is still an area which has high educational value. It is no longer uncommon for universities to offer courses and perhaps even majors in the field of game development Finkel et al. [8]Argent et al. [4]. Schaefer and Warren [23] provide a guide for teaching students about geometric modeling and computer graphics through game design. Many open source games engines and languages exist such as the Glest engine[2] which makes a perfect candidate for use in the scene-to-adventure game system which will be discussed in Section 2.4.1.3.

2.4.1.2 Non Programming Based Development

The development of non-programming based game development tools is responsible for a large surge in the number of independent games currently available. These tools can also be used to speed up the game development process for existing game developers.

Torrente et al. [28] provide an open source environment, the <e-Adventure3D>, for the creation of 3-dimensional educational adventure games. The application Torrente et al. [28] simply consists of an authoring tool and game engine giving those users with no programming knowledge the ability to create games. A noticeable part of the system Torrente et al. [28] is that the storyboard for the created games uses an XML-based language and modifying the game becomes as simple as editing certain variable values. This use of XML has potential to be largely useful in the development of a scene-to-adventure game system and is discussed further in the next section.

Many other systems exist which allow users to create games without using any programming language such as GameMaker[3] and Adventure Game Studio[4]. These systems offer development techniques such as a drag-and-drop interface and generic objects which prove that large parts of game development can be generalized. This is of great importance in developing a text-to-game system

[2]http://www.glest.org/en/engine.php
[3]http://www.yoyogames.com/gamemaker
[4]www.adventuregamestudio.co.uk

```
<unit>
        <parameters>
                <size value='#' />
                <height value='#' />
                <max-hp value='#' regeneration ='#' />
                <max-ep value='#' regeneration ='#' />
                <armor value='#' />
```

Figure 2.2: Defining Properties for a custom unit in Glest

2.4.1.3 Markup Language in Games

The creation of non programmer based game development tools has become increasingly popular. Much effort has gone into creating systems which will allow the creation of both simple and advanced games with minimal effort. In order for this to be achieved, it was necessary for developers to create generic interfaces to make game engines accessible to those unfamiliar with programming. Sepchat et al. [24] created a semi-automated game creation system which creates tactile games for visually impaired children. While the scope of such a system is significantly smaller than that of the potential scene-to-adventure game system, it is interesting to observe the interface used as it makes use of generic properties which can be tweaked to alter the game.

One way of accomplishing this is through the use of a markup language such as XML. Simply filling in meta data is a much easier process than learning to program. Many of the systems mentioned above such as the <e-Adventure3D> Torrente et al. [28] and the Glest engine [5] make use of XML in the same manner. An example of the typical use of XML in the Glest engine can be seen in Figure 2.2[6]. Moreno-Ger et al. [19] developed a system similar to <e-Adventure3D> Torrente et al. [28] which also makes use of a markup language for the story board and a processor (game engine) for the language.

These markup language interfaces prove to be very useful in the development of a scene-to-adventure game system as XML is used to a large extent in both text-to-scene system and several modern game engines.

[5]http://www.glest.org/en/engine.php
[6]http://glest.wikia.com/wiki/GAE/Unit_XML

2.4.2 Automated Game Development

Automated game development is still a relatively small field. However, work is being done on creating systems which automate the design process for games. Nelson and Nelson and Mateas [20] developed a prototype system which is aimed at the automatic creation of small "WarioWare" style games[7] such as clicking on a fast moving object in a small screen. The main tasks are building the system with the ability to make sense of the abstract rules for the games and determining some methods for visualizing these rules. Nelson and Mateas [20] also faced the problem of dealing with common sense as the system worked by allowing users to specify verbs or nouns to describe the game. Another attempt at automated game creation was made by Togelius and Schmidhuber [27] in which the initial game was generated and left to evolve. This involves the creation of neural networks to guide the evolution of the NPC (Non-player character) [8] controllers as well as the game rules Togelius and Schmidhuber [27]. However, the paper does not contain results and it is therefore difficult to imagine the possible current applications of such a system.

While these systems have potential in opening a new field of computer science, the methods which they use are only applicable to a very limited set of small games. The development of a scene-to-adventure game is a far more plausible concept with the currently available technology as the story and narrative structure provide a basis for the goals and events of the game.

In this section we have shown that there is a clear link between the input of several of the available game development tools and the annotated text which modern text-to-scene systems generate in creating their output.

2.5 Summary

These 3 significantly different research areas provide the resources needed to create a working Text-to-Game system. Parts-of-Speech taggers and Text-to-Scene systems have all of the functionality required for the initial step of annotating the fiction text efficiently and with a usable level of accuracy. Relating directly to this is the structure which many modern games are taking by creating generic interfaces for ease in game development. With this functionality available we show how the annotations acquired from the first step can be used in providing the initial detail

[7]Small and simple mini-games.

[8]A computer controlled actor such as an opponent.

for the games. This can then be used in crafting enjoyable and playable games with rich environments and narratives based on popular fiction texts. Another important aspect of this survey is the design of text-to-scene systems. Separating such a system into 3 main components, namely the environment characters and events, provides a good starting point for the design of our Text-to-Game system.

Chapter 3

Design

3.1 System scope

The concept of a Text-to-Game system is best thought of as a process. Like most systems it is typically concerned with taking input, modifying that information in some way, and producing output. However, while Text-to-Scene systems such as that presented by Glass [9] deal with the annotation of text, we are not concerned with this part of the process. Instead, we make use of predefined annotations to give us the appropriate information meaning that any text is annotated by hand. It should be noted that the story concepts which we use in the creation of the system are those typically found in Adventure/Fantasy texts as they are best suited for creating an enjoyable game. Figure 3.1 illustrates the scope of the system. The actual components of the system will be described in the next chapter.

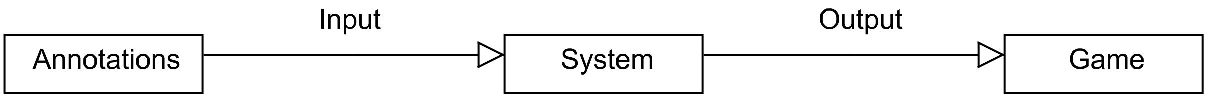

Figure 3.1: Overview of the system scope

3.2 System Components

Section 2.2.2.2 describes how existing Text-to-Scene systems exploit the idea that most texts can be broken down into an environment, characters and events in their design. We adopt this approach in designing our Text-to-Game system as we create our components to accommodate these 3 key elements. Following sections in this chapter detail how the system was designed in relation to these 3 elements. However, while these system components are crucial for the creation of a Text-to-Game system. Implementation would not be possible without a game engine to support the creation of the game. The next section describes the game engine used and how it relates to the system.

3.2.1 The Game Engine

We make use of the Glest engine mentioned in section 2.4.1.1 as its extensive use of XML is ideal for the text-to-game system. However, while this engine is suitable for the needs of a Text-to-Game system, the implementation of the system components are only applicable to the Glest engine. This means that switching to another game engine would require major refactors of the system components. However, by showing that the system is dependent on the game engine we generate another potential problem. This problem is whether or it is possible to generate a game engine

An important aspect of the engine which plays a large role in our system is that of scenarios. Scenario are used to store all of the global information for a particular game. For example, the characters to appear in the game, the name of the map and the goals are all stored in the scenario. We make extensive use of this as it allows for many game to be stored centrally and share the same resources (3D models, sounds, images) therefore conserving physical memory.

The game also has the necessary provisions to aid the creation of those 3 key components mentioned in section 2.2.2.2. However, these will be described in following sections. The next section describes how part of the system was designed to handle the creation of the environment in the game.

3.2.2 The Environment

Typically video games make use of maps to represent the environment. Therefore, the words environment and map may be used interchangeably. In determining the key features of the environment we examine an illustration of a map depicting the world from the Lord of The Rings[1] series of fantasy texts. We discover that there are 2 main elements to such an environment. Those being points of interest and paths.

[1]Popular fantasy text written by J. R. R. Tolkien.

Points of interest are those places in the environment which have some significance in the story. For example, a point of interest may be a town or a major geographical feature such as a lake or mountain. In order to create links between these 2 places we need paths such as roads and footpaths.

In order to allow the system to create environments we require a map generator capable of converting the appropriate annotations into an environment.

3.2.2.1 The Map generator

In order to create maps with the given annotations we have also created a map generator. This is one of the major system components and is simply responsible for generating a map given the environment annotations. Figure3.2 shows the relationship between the annotations and the map.

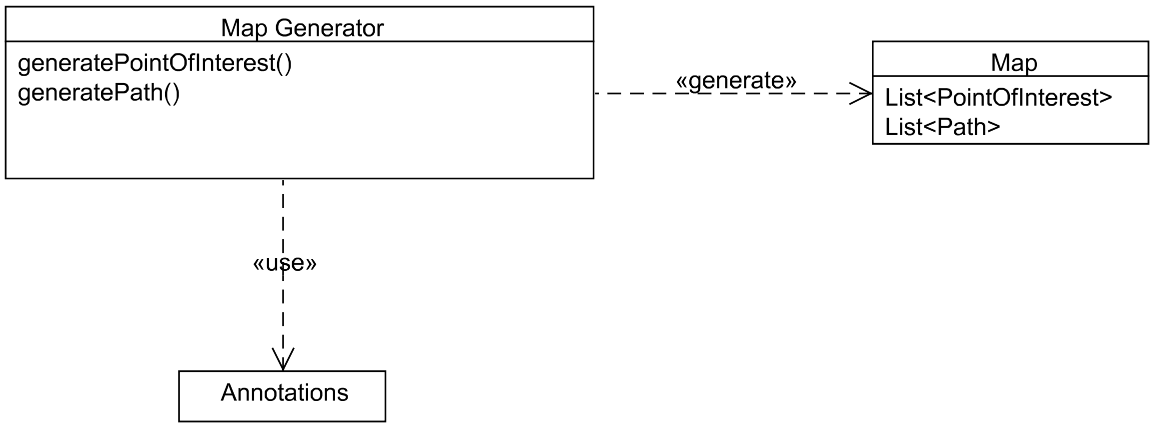

Figure 3.2: Overview of the map generator design

3.2.3 The Characters

In designing how the system handles characters it was necessary to examine what character information would be available in the text. It was also necessary to consider how characters could fit into the game. Typically characters in games are either controlled by the player or the computer. Also characters in games are typically separated into teams which have opposing goals. Also, many fantasy texts have many races (for example, trolls, humans and elves). Fortunately, strategy games such as Glest make use of factions to separate characters into races or teams.

Another important aspect of the characters is their appearance. However, generating the 3D models and textures for each character is outside of the scope of the project.

Typically Text-to-Scene systems such as Wordseye Coyne and Sproat [6] and that created by Zhu et al. [30] make use of an existing database of images or models. We take the same approach by storing a set of 3D models and texture which can be used for the characters appearance.

Along with this we realise that the spatial relationships between characters in a book is important and so characters need starting positions which are relative to the game map.

Examining this information we define the initial properties which characters will have. These include a name, a faction, a control method (computer or human), a model (appearance) and a starting position . In order to structure these characters into their factions appropriately we make use of the concept of a character tree. Figure 3.3 illustrates the structure of an example character tree.

The next section describes how events will be recreated in the game.

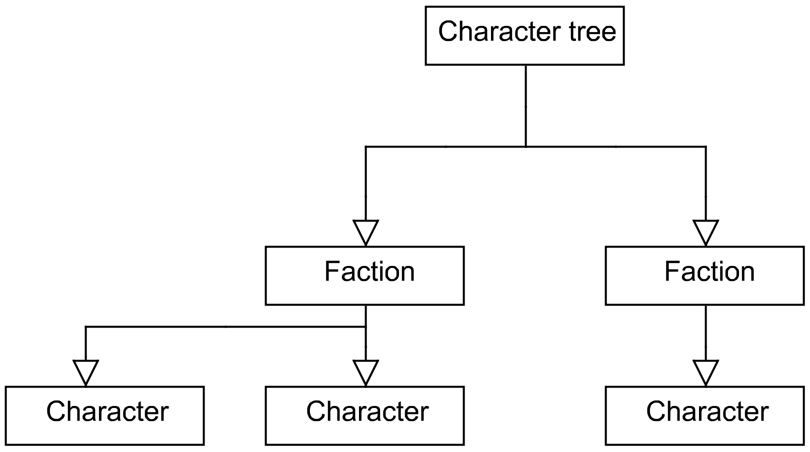

Figure 3.3: Example of the character tree structure

3.2.4 The Goals

In creating the game we wish to recreate any events described by the text. However, while one of the goals of the system is to accurately represent the events of the story, we need to tailor those events to fit the playable game. For this reason, we describe a way of converting those events which involve the playable characters into goals which the player can complete. In order to achieve this it was necessary to create a standard way of representing multiple different types of goals. Examples of such goals would be the character needing to reach a specific destination or collecting an item before a specified amount of time passes.

Goals or objectives in games can be thought of as conditions which the player must meet. Also, we consider that the completion of a goal often leads to some action such as a reward for the player. We therefore initially consider a goal in terms of a condition and action. However, we also need to know when to check these conditions to see if they are met. We therefore introduce the concept of a trigger. Triggers specify an action or event which must occur in order for a goal to be completed. This allows us to know which conditions should be checked and when. Having defined these 3 concepts we now consider goals in terms of triggers, conditions or actions.

While designing the goals and characters is an important part of the creation of the system, a mechanism for generating these elements must also be discussed. The next section describes the game generator which is responsible for converting the character and goal annotations into their virtual forms.

3.2.5 The Game Generator

The game generator is the system component responsible for converting the annotations into the characters and goals. However, this is not the game generators only function. Section 3.2.1 describes the concept of a scenario. Another important function of the game generator is the generation of the scenario file. once the map, characters and goals have been generated, the scenario file is created and filled with the global game information. Figure 3.4 shows an overview of game generator in relation to other relevant parts of the system.

The next section explains the overall system design and how the various components fit together.

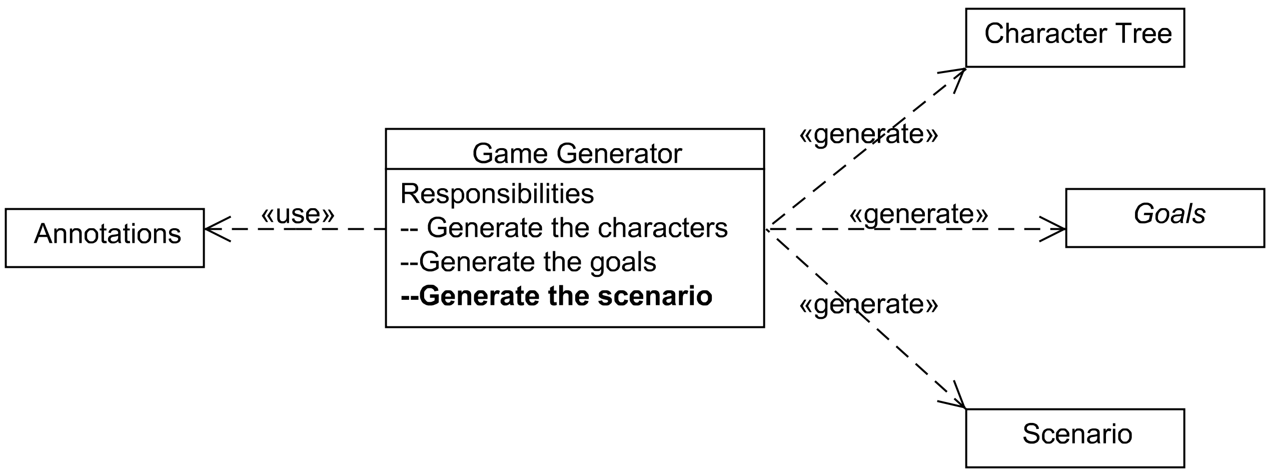

Figure 3.4: Relevance of the game generator within the system.

3.3 System overview

While accurately generating the map, characters and goals is a necessary part of the Text-to-Game process. It is also important that the system is structured in such a way that allows these elements to eventually be combined to create a game. The last system component which we define is the actual Text-to-Game application. This will simply be responsible for taking in the annotations and calling the map and game generators to create the game. Figure 3.5 shows how all of the components and elements mentioned in this section fit together to create structure of our Text-to-Game system.

3.4 Summary

This section has discussed our design choices for recreating those key story elements in the game. By considering each of these elements both in terms of the text and the game we have created a system design capable of supporting our Text-to-Game system. The next chapter discusses how we have implemented these system components and provides a lower level explanation of how the system functions.

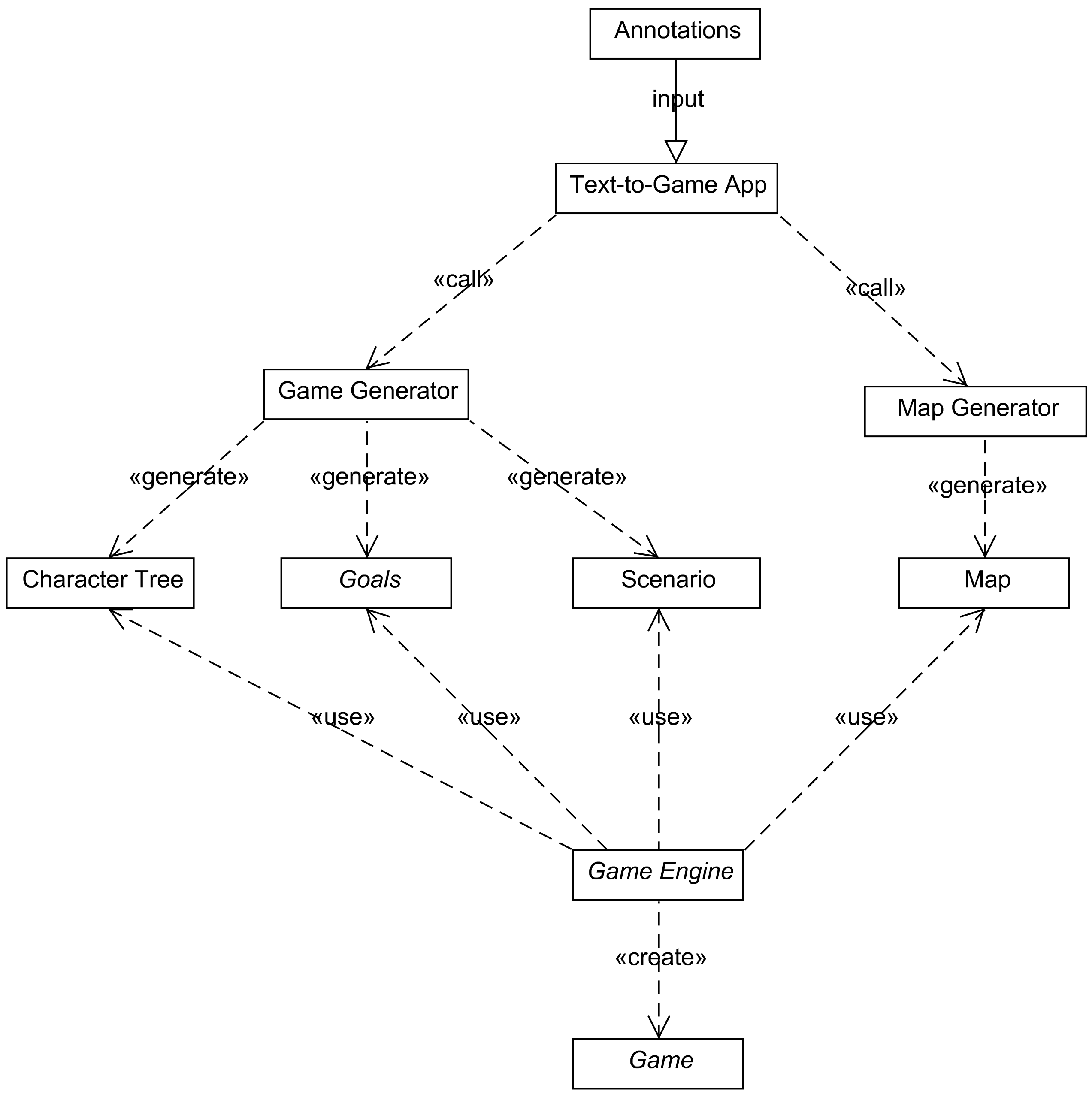

Figure 3.5: System structure overview

Chapter 4

Implementation

In this section we describe how the concepts and system components mentioned in Chapter 3 are implemented. However, we must first discuss the annotations in context of the system. Also, we describe the facilities offered by the Glest engine as these are used extensively throughout the system.

4.1 The Annotations

With the various game components come many different annotations. For this reason we require that annotations are stored centrally to avoid confusion. Therefore, we have created the concept of a Game Descriptor File (hereafter referred to as the GDF). The purpose of which is simply to store all of the annotations in the same place. This means that whenever a component of the system requires some annotation information it must parse the GDF to find what it is looking for. Figure 4.1 shows an outline of the GDF and all of the possible annotations.

The next section discusses the important facilities offered by the Glest engine.

4.2 The Game Engine

As Section 3.2.1 describes, implementation of the system components is relative to the game engine. This section details the facilities made available by the Glest engine as these are important aids in implementing the various system components. The 3 main facilities provided by the engine are the Map class, the tech tree and the scenario file. We will begin by examining the Map class. However, before this is done a brief explanation of the use of XML in the Glest engine is required.

```
<Game name>
        <Map name>
                <Regions>
                        <region name radius xcenter ycenter heightmin
                            heightmax jaggy object surface />
                        ...
                </Regions>
                <Paths>
                        <path start end />
                        ...
                </Paths>
        </Map>
        <CharacterTree name>
                <faction value type />
                ...
                <character name faction startingposition model/>
                ...
        </CharacterTree>
        <Goals>
                <Goal>
                        <Trigger></Trigger>
                        <Condition></Condition>
                        <Action></Action>
                </Goal>
                ...
        </Goals>
</Game>
```

Figure 4.1: GDF skeleton.

4.2.1 XML in Glest

Almost any element in the Glest engine is defined with an XML file. This includes, factions, characters (referred to as units in the Glest engine) and scenarios. Also, some of these elements have additional resources which are all declared in the XML definition file. For example, units in Glest consist of a sound folder which stores sounds, a models folder which stores the 3D models and the units XML file which defines all of the units properties.

We will now look at the first important facility offered by the Glest engine, the Map class.

4.2.2 The Glest Binary Map

The maps which the game engine makes use of are stored using a Glest Binary Map (.gbm) format. Map are stored as 2D array of cells, each of which has several properties for defining the cell. Figure 4.2 shows the initialisation of the cells and the information contained in each cell. Another important tool offered by the Glest engine is the map editor (pictured in Figure 4.3) which allows for visual creation of maps and is described in the next section. The map editor offers an overhead view of the map and allows users to click on cells to change their properties. This also introduces a useful concept which is the abstraction of brushing values onto the map. Instead of simply editing one cell, users can change the 'brush' size and alter all of the cells in a circular area (the brush tip) at the same time. This concept is used extensively in the map generator portion of the system and will be explained further in section.

The next important facility offered by Glest is the tech tree. This will be discussed in the section below.

```
Cell ** cells;

struct Cell{
        int surface;    //texture drawn onto the map
        int object;     //the object (building, tree) placed on the map
        int resource;   //the resource (gold, stone) placed on the map
        float height;   //the height of the map
};
```

Figure 4.2: Creation of the cell struct and cells array

4.2.3 The Tech Tree

The tech tree in Glest is a structured collection of files and folders which is used to separate the units (characters) into factions. A single tech tree can have any number of factions which in turn can have any number of units. This is highly important in our system as it offers the perfect facilities needed to create a character tree. What should also be noted is the upgrades which factions have. While factions are primarily for storing units, they also allow for upgrades. These are simply modifications which can be applied to individual units in the faction. Upgrades are worth mentioning as they provide a method for modifying characters after the game has started. The next section describes the concept of a scenario within the Glest engine.

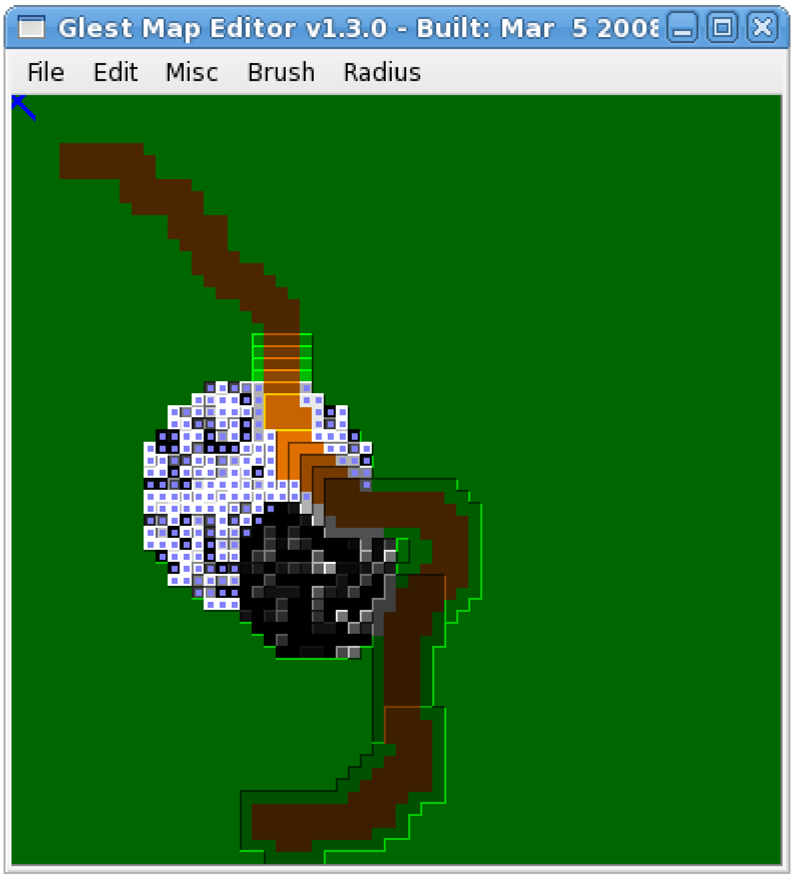

Figure 4.3: Screen capture of the Glest map editor

4.2.4 The Scenario

As mentioned in Section 3.2.1, scenarios are a means for storing the information about a particular game centrally. All of the details describing a particular game are stored in the scenario file. Figure 4.4shows an outline of the scenario file while Table 4.1 provides definitions for each of the properties found in the scenario file. It is worth taking note of the scripts section as this is where the goals and events for games are stored. The Glest engine offers the facilities to create Lua scripts which can be called from within the glest source code.

The next section describes the first major system component, the map generator.

```
<scenario>
        <difficulty value/>
        <players>
                <player control faction team/>
                ...
        </players>
        <map value/>
        <tileset value/>
        <tech-tree value/>
        <default-resources value/>
        <default-units value/>
        <default-victory-conditions value/>
        <scripts/>
</scenario>
```

Figure 4.4: Scenario file skeleton.

Property	Purpose
difficulty	Determines how difficult the game is. For example, a more difficult game may have stronger enemies.
player	A character, or team of characters which will feature in the game.
map	Specifies the virtual world in which the game will take place.
tileset	The tileset used will determine the type of scenery found in the game. This is purely for aesthetic reasons.
tech-tree	Specifies the name of the character tree which will be used in the game.
default-resources	Used to toggle whether or not the various teams start with a default value of resources which is specified for each faction.
default-units	Used to toggle whether or not the various teams start with the default units specified for each faction.
default-victory-conditions	Used to toggle whether or not the predefined default values for the victory conditions will be used.
script	Each (lua) script represents a different trigger which may be used in specifying goals for the game.

Table 4.1: Scenario property definitions.

4.3 The Map Generator

As mentioned in Section 3.2.2, the creation of a map depends on the ability to create both points of interest and the paths between them. Therefore, the first task in creating the map generator was to define an abstract way of representing the

```
class Region {
        public:
        Circle brush;     //The size and coordinates
        double heightmin;//The minimum height
        double heightmax;//The maximum height
        double jaggy;     //The amount of variance in height between cells
        int object;       //The object placed onto each cell in the region
        int surface;      //The surface drawn onto each cell in the region
        int resource;     //The resource placed onto each cell in the region
    int density;      //The density of the object within the region
};

class Circle {
        public:
        float x;          //The x coordinate of [the center of] the region
        float y;          //The y coordinate of [the center of] the region
        float r;          //The radius of the region
};
```

Figure 4.5: Region class information

points of interest. The next section describes how we create such a representation.

4.3.1 Regions

In order understand the possible ways of representing points of interest we consult the map class. As mentioned previously the 2D array structure of the the map allows us to draw elements onto the map more than one cell at a time. Therefore we recognise that it is possible to change the cell properties of entire areas on the map. However, if we consider the complexity of drawing random shapes versus the complexity of drawing circles we realise that much time and effort can be saved by making use of circles to draw elements onto the map. This led to the creation of the concept of a Region.

We define a region as being a circular area with certain properties. Figure 4.5 shows the properties of the Region class. As we can see the Region class has all of the information contained in the cell struct along with a Circle (consisting of a radius and coordinates) and some other values. This class allows us to create circular areas with any object. The next section describes how these properties are used to create regions on the map.

4.3.2 Creating Regions

In order to draw Regions onto the map it was necessary to create a makeRegion() method. This works by looping through all of the cells in the map and checking if the they fall inside the region. If a cell does fall inside the region, its height, surface, object and resource are changed accordingly. Changing cell properties is done with a respective method in the Map class. For example, the method changePointHeight(x, y, height) will change the height of the cell specified to the x and y coordinates.

While this method is efficient for drawing regions, we do not always want the height of the cells in a region to be the same. Therefore, we make use of the procedural terrain generation techniques created by Alcock [3] to calculate realistically varying heights. This allows us to create a range of geographical features including flat fields, rolling hills, lakes and mountains. The minimum and maximum heights of a region are modulated by the respective properties in the region class while the extremeness in the difference between the heights is altered by the 'jaggy' property. Figure 4.6and 4.7show samples of the regions created with the makeRegion() method.

Another important aspect of the the method is the ability to create regions with varying object densities. This is done with a roulette wheel type selection in which a random number is generated between 1 and 100. If the number is less than or equal to the density property of the region it will be placed on the current cell. This is an important facility especially when creating towns as the number of buildings per an area of size n would be far smaller than the number of trees in a forest of size n.

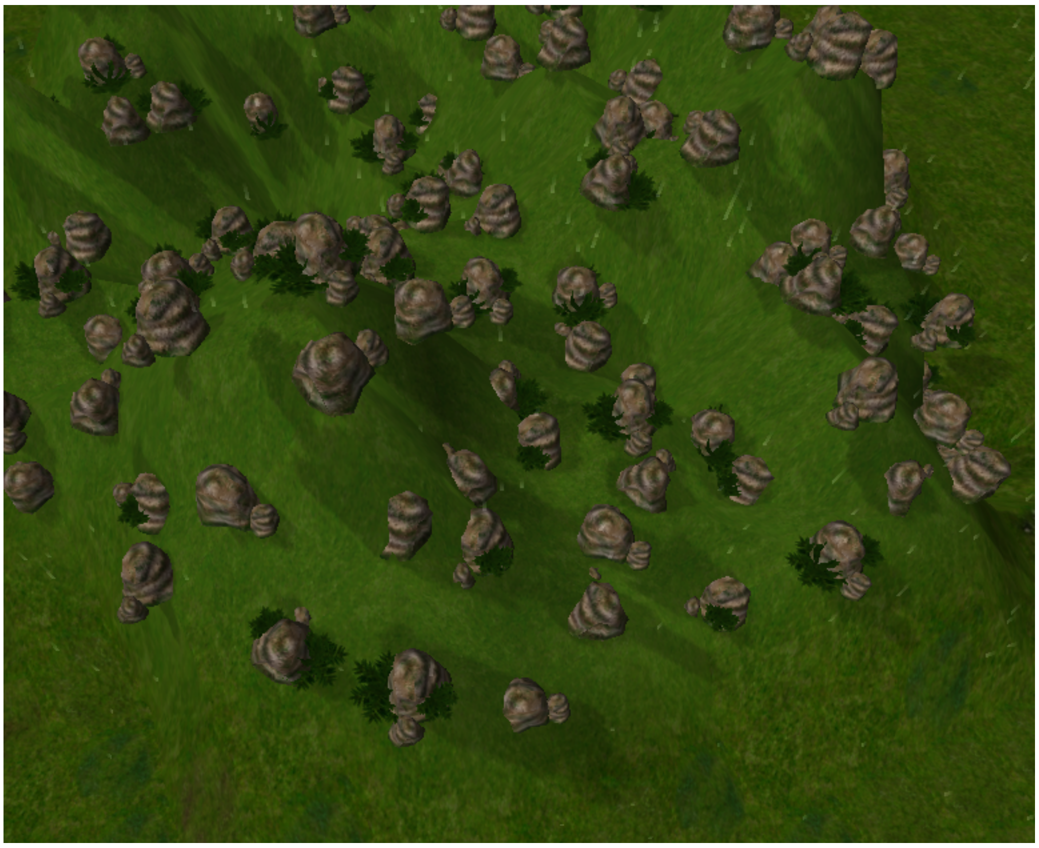

Figure 4.6: Region Example 1

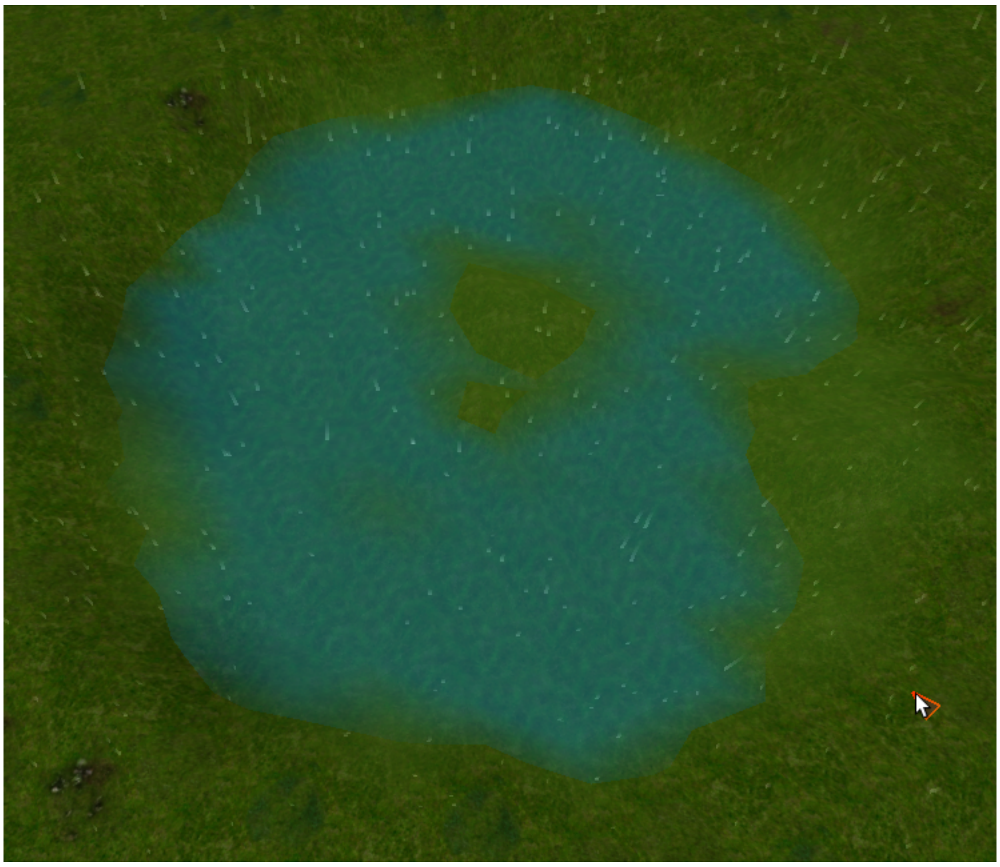

Figure 4.7: Region Example 2

We shall now look at concepts of paths in relation to the system.

4.3.3 Paths

Paths are simply described as being a graphical guide between 2 points. Therefore creating a path could be as simple as changing the texture of all of the cells between 2 points. However, this would create only straight paths. The next section describes the techniques we use to create realistically formed paths.

4.3.4 Path Creation

Creation of the paths is done with a recursive subdivision algorithm. This means that the midpoint between the start and endpoints is calculated and a path is created between the start and middle points and another between the middle and end points. However, the surface texture is only changed where the length of a path is less than or equal to 1. Also, once a midpoint has been calculated, a horizontal and vertical offset the size of which is determined by the paths 'jaggy' property. This is added to the position of the point. The size of the offset is depended on the depth of recursion meaning that large offsets will only be seen towards the centre of the path. These offsets help in creating paths realistic paths which are not exactly straight.

Another key feature of the Path class is the smoothing algorithm. Prior to implementation of the smoothing algorithm a path which went over a highly jaggy mountain would be unwalkable. An example of such a path can be seen in Figure 4.8. The smoothing algorithm works by assigning the height of a given point, the average of heights of surrounding points. The result of enabling this smoothing algorithm can be seen in Figure 4.9.

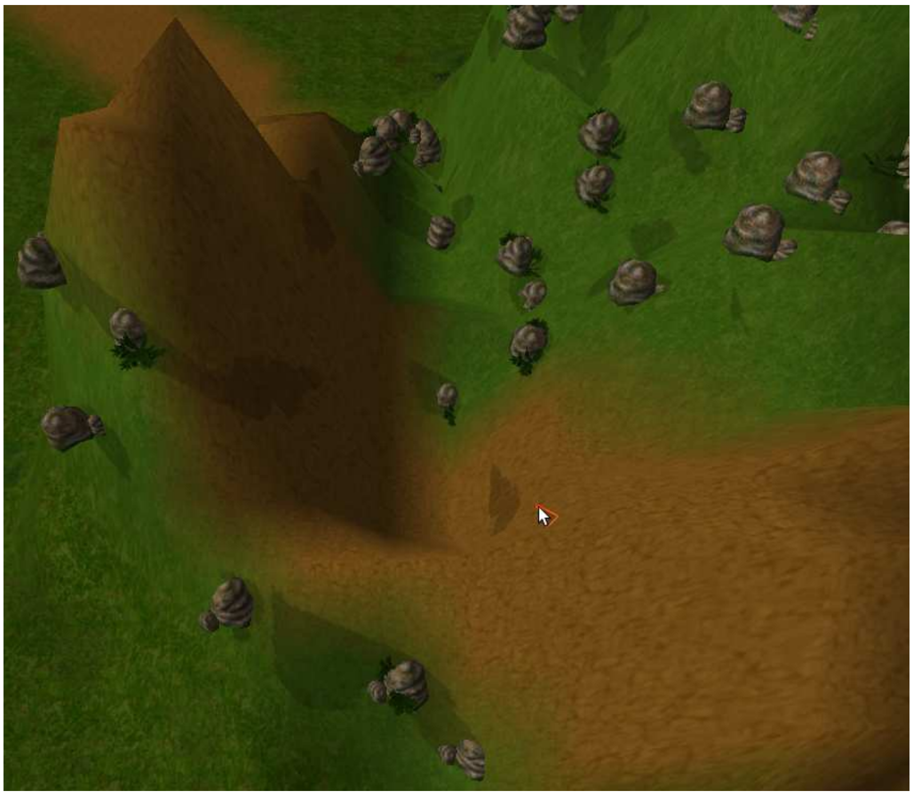

Figure 4.8: Unsmoothed path running over extremely jaggy terrain

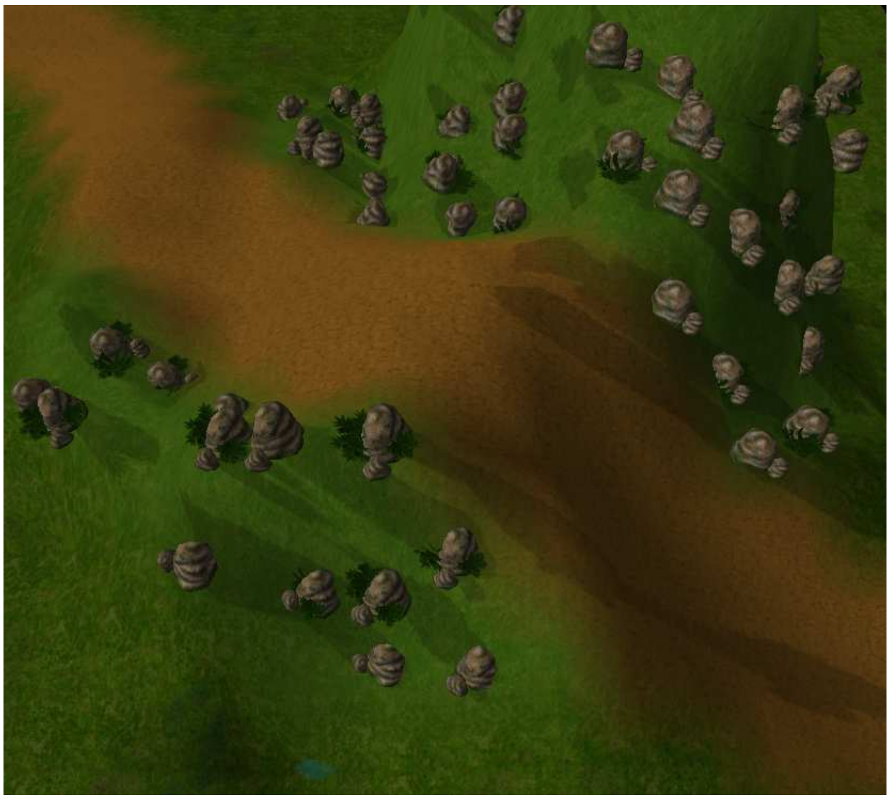

Figure 4.9: Terrain smoothed by path smoothing algorithm

The map generator is solely responsible for the creation of the map. This is unlike the game generator which is responsible for the creation of all of the other elements of the game. This includes that character tree and the goals. The next section discusses how the game generator is implemented to create he character tree and goals.

4.4 The Game Generator

As mentioned previously the game generator is responsible for the creation of the character tree and the goals. Another important task of the game generator is the creation of the Scenario file as this is essentially what describes the entire game. While the game generator is responsible for producing a large part of the game, it should be mentioned that it is simply a python script which works by parsing the GDF and producing the relevant game elements as well as an entire scenario file which is discussed in the next section.

4.4.1 Creating the Scenario

Figure 4.4 provides an outline of the scenario file. Many of these these properties such as the difficulty and default-victory-conditions are generated with default values. However, values such as the faction name and player nodes are created as the the various components of the system are generated.

The next sections discuss how the character tree is generated.

4.4.2 Creating Factions

The file structure of Factions can be seen in Figure 4.2. This consists of upgrades, music, characters and an XML file which describes the faction. Values in the XML file specify faction-specific characteristics such as the different armor classes and attack styles which the units in the faction will be able to use. In order to simplify the process of creating a faction, we do not modify the XML file for each faction, instead we simply use a default faction XML file which is copied to the faction directory and renamed with the faction name. We also use default music as this removes the need for content generation which is not within the scope of this project. Once the factions have been created, character can be created and placed in the various factions. How this is done is explained in the next section.

4.4.3 Creating Characters

Characters in the game engine consist of a sounds, models, images and an XML file (hereafter referred to as the descriptor) as illustrated by 4.2 which describes the characters properties. In order to understand how characters can be generated, we must consider the annotations which describe a character. As mentioned in section 3.2.3 the character properties which we are concerned with are a name, a faction, a control type, a starting position and a model. The starting position and control type will be stored in scenario which is described later and we can therefore ignore these 2 properties for the time being. The property which we are most concerned with now is the model. The reason being that the model does not simply specify the appearance of the character. It also specifies what skills the character will have and various other attributes. The reason for this is that a characters descriptor describes all aspects of the unit, including its appearance and sounds. Therefore, creating a unit is simple as it involves nothing more than copy the unit files of the appropriate model into the faction folder and renaming the characters descriptor.

However, issues where encountered initially as a characters descriptor defines skills or attributes which may be faction specific. Therefore, errors were received where the characters descriptor made use of game elements such as an upgrade or armor type which was not defined in the faction. In order to prevent this, the descriptor of any model which is added to the system must be checked for faction dependent properties which must then be removed.

The next section describes how goals are created.

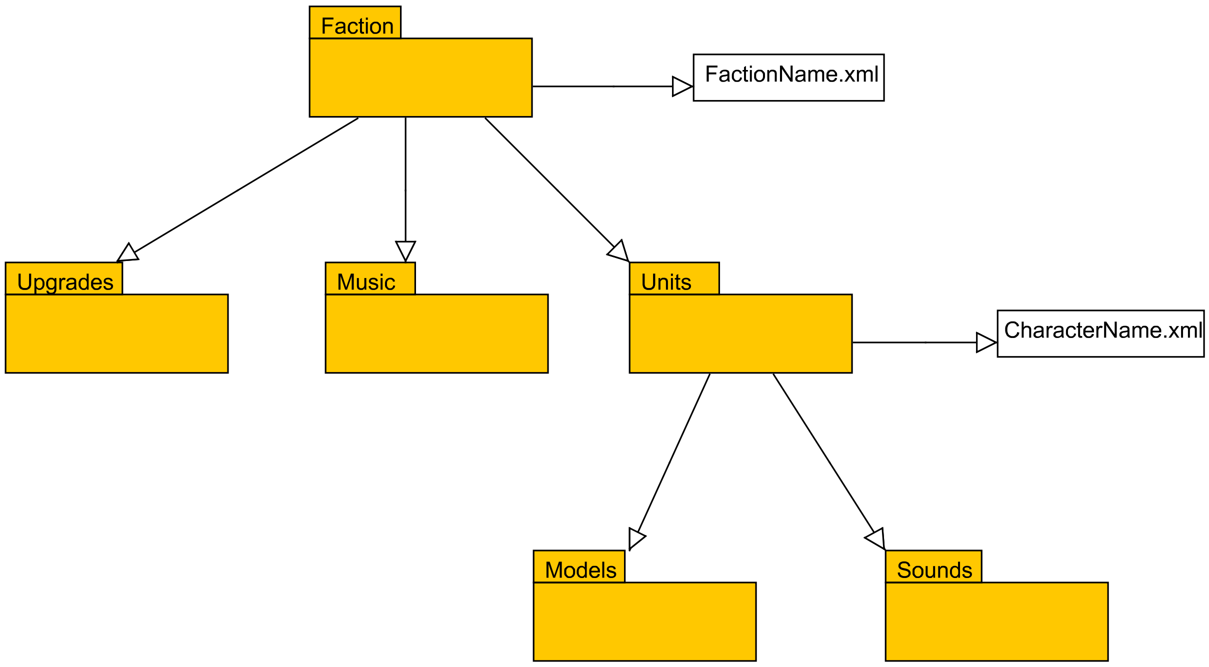

Table 4.2: Character Tree file structure

4.4.4 Creating the goals

In section 3.2.4 we define goals in terms of triggers, events and actions. As mentioned in section 4.2.4 the goals are stored in the scenario file. Glest makes provisions for these goals by providing a (Lua) scripting interface. To best understand how we have implemented goals, we discuss the following example. Consider the following text: “Tramus needed to reach Wiseman Pat in order to learn about the about the dangers which approached”. The trigger would be Tramus moving, the condition would be Tramus being near to WisemanPat and the action would be Wiseman Pat informing Tramus. The annotations which we define for such a goal are based structured to accommodate the Lua scripts used by the engine. The resultant annotations can be seen in Figure 4.10. In this example, unitMoved, unitNearUnit and showSimpleMessage are all examples of lua scripts which the system is capable of understanding. Figure 4.11 shows what the goal will look like when it is placed into the scenario file by the game generator. As we can see the unitMoved or trigger script has been placed in angle brackets. This shows that it is a script which will be called when a certain event has happened. Also, the condition has been wrapped in an if else statement and an 'end' statement has been placed after the action.

```
<goal>
        <trigger>unitMoved</trigger>
        <condition>(unitNearUnit(Tramus,5, WisemanPat) == 1)</condition
           >
        <action> showSimpleMessage(You must now journey to the East to
            find the golden sword, Hello Tramus)
        </action>
</goal>
```

Figure 4.10: Goal annotations

```
...
<scripts>
        <unitMoved>
                if (unitNearUnit(Tramus,5, WisemanPat) == 1) then
                        showSimpleMessage(You must now journey to the
                            East to find the golden sword, Hello Tramus
                            )
                end
        </unitMoved>
</scripts>
...
```

Figure 4.11: Goal annotations

Chapter 5

Evaluation

5.1 Goals

In designing the evaluation we consider the high level functions of the system. Those being the ability to act as a game development tool and recreate the characters, events and environment described in a piece of text or story in the form of a game. We therefore decide that evaluation of the system must achieve 2 goals. Firstly, we must discover how accurate the game is in relation to the story and secondly, we must determine the quality of the produced game.

However, there is no quantifiable method for evaluating how accurately a game represents a story. Also, there is difficulty in evaluating the quality of games due to a lack of standard quantitative review criteria. The next section details how we overcome these issues and describes the design of the system evaluation.

5.2 Methodology

While the system has been developed as a game development tool, the games which it creates will ultimately be played by the end-users. For this reason evaluation of the system takes place in the form of user evaluation. In order for users to evaluate the system properly, it is necessary that they evaluate the game not just as a game, but also as a product of the system. This requires that users are able to play the game and then relate it to a story. To achieve this a short story was written and then used to generate a game. Section 5.2.1 describes how the story was written

In order to record results, a questionnaire was drawn up with several Likert statements and open questions which will be described in following sections.

5.2.1 The Story

While the primary goal of the system is to show that it is possible to implement a Text-to-Game system. Complexity is introduced by the fact that text can be used to describe any number of events, characters or environments. It is therefore necessary to bear in mind the scope of the system. Advanced features such as generating models for the appearance of characters is not within the scope of the project. While it is possible to manually add support for new character models, environmental details and goals we did not feel it was necessary as this was not within the scope of the system. Therefore the characters, events and environment in the story are confined to the content which is currently offered by the system. The story can be found in Section A.1 of Appendix A.

The next sections describes the design of the questionnaire.

5.2.2 The Questionnaire

As mentioned in Section 5.1, the purpose of this evaluation is to ascertain the level of accuracy with which story is represented by the game and to determine the quality of the game. The sections below explain how the questionnaire was designed to reflect this.

The questionnaire can be seen in Section A.2 of Appendix A.

5.2.2.1 Measuring game Accuracy

Initially the questionnaire drawn up had only 1 Likert statement relating to the accuracy of the game. Users were asked if they felt that the game was an accurate representation of the story. However, after several test evaluations we found that this did not provide enough detail. In order to fix this issue we consider how the relationship between the game and story can be defined in terms of the environment, the characters and the events. Therefore the Likert statement relating to accuracy was replaced with 3 newer statements relating to the map, characters and goals (Questions 1 to 3 in the questionnaire).

5.2.2.2 Measuring the Quality of the Game

As mentioned in Section 5.1 the difficulty in evaluating the quality of a game is introduced by the face that there is no standard for doing so. Many of the game reviews found on the Internet make use of several categories to judge each game on a standardised criteria [1, 21]. While different reviewers may use similar categories, there is still no standard for defining these categories. Beale and Bond [5] attempt to create such a standard by performing a grounded theoretical analysis. They describe grounded theory as an approach to research in which hypotheses about some scenario are formed from raw data. More specifically, [5] compare several different game reviews in order to determine which characteristics make a "good" game and which are most common in "bad" games.

The results obtained show that cohesion, variety, good user interaction and a social aspect are the characteristics which are responsible for a good game while technical soundness, customisable and a good environment are some of the characteristics of moderate importance. However, variety and cohesion can only be seen in significantly large games while the game used for evaluation is a small game, only lasting several minutes. Also, the social aspect is not within the scope of this project and is therefore not relevant.

While these characteristics may be necessary to create good, commercial games. The criteria by which we choose to rate our games must accommodate the fact that they are products of a Text-to-Scene system. Therefore, along with those Likert statements which measure the accuracy of the game, we provide 2 additional [fairly broad] questions for measuring the quality of the game. These relate to technical soundness and the control scheme and interface of the game (Questions 4 and 5). Additionally, we include an open question in which users are asked to mention any improvements to the game which they could recommend.

The next section describes the experimental process which was performed during the testing period.

5.2.2.3 The Process

Candidates were obtained on a volunteer basis and tests happened individually and in isolation of the other candidates. Participants where first asked to read the story found in Section A.1 of Appendix A. On completion, the game was started and candidates were given a short tutorial on the control scheme. Candidates were then left to complete the game until they won or lost. Candidates who lost the game were asked to replay it until they won. This was to allow candidates to experience the entire game. On completion of the game participants were handed a copy of the questionnaire and asked to complete it in their own time. Once 10 questionnaires had been returned the results were examined.

The results obtained are shown in the next section.

5.3 Results

Due to the structure of the questionnaire both quantitative and qualitative results were obtained. Table A.1 of Appendix A shows the scores obtained from the questionnaires where 5 is the highest (best) score and 1 is the lowest. Candidates where also asked to give reasons for their scores thus allowing us to gain insight into the results. Going back to the problem statement, the major goals of the system are to generate the characters, environment and events described in text. Therefore, examination of the results will happen in separate sections according to these elements. We also examine those results relating to the gameplay and the game engine separately to allow us to examine the quality of the game. We begin by examining the results for the first question, the accuracy of the map.

5.3.1 Accuracy of the Map

The results for the map portion of the evaluation can be seen in Table 5.1. The high mean and mode are good indications that users felt the map accurately represented the environment in the text. Several users mentioned that by following the directions mentioned in the story they were able to navigate to the various points of interest. It was also mentioned that developers have different interpretations of the story map and it is therefore important that directions in the game match that of the text. Directions in this case being the spatial orientations of points of interest. For example, if a town lies to the east of some trees, this should be accurately recreated in the game.

The next section will look at the results obtained for the characters.

Scores:

Run	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Score	4	4	5	4	4	5	4	5	5	5

Additional statistics:

Mean	4.5
Median	4.5
Mode	4
Maximum	5
Minimum	4
Range	1

Table 5.1: Results for the map portion of the evaluation.

5.3.2 Accuracy of the Characters

Table 5.2 summarises the results obtained for the character section of the questionnaire. While the mean is still above 4, we must examine the existence of several low results. By examining test 2 and 5 which resulted in low scores of 3 we find some insight into the reason for these lower scores.

The reason for the score in Test 2 as described by the participant is: “Its hard to gather much into about the characters from the story”. This shows that determining if the characters in the game accurately match the characters in the story may only be possible if characters in the story are described with a certain level of detail. An issue which might be fixed by increasing the level of detail in the story. The reason for the low score in Test 5 as described by the participant is: “Suppose [rating the character accuracy] is interpretational. I viewed the deamon differently”. This introduces another issue with evaluating the character which involves the characters appearance. Different readers imagine the appearance of objects, environments and characters differently. Once again, we see another reason to increase the level of detail provided by the story as this would limit the amount of information which would be imagined by the participant. The next section discusses the results obtained from evaluation of the events.

Scores:

Run	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Score	4	3	5	4	3	5	4	4	4	5

Additional statistics:

Mean	4.1
Median	4
Mode	4
Maximum	5
Minimum	3
Range	2

Table 5.2: Results for the map portion of the evaluation.

5.3.3 Accuracy of the Events

Table 5.3 summarises the results obtained for the character section of the questionnaire. Once again a moderately high score is achieved. However, user comments in this section of the questionnaire were scarce. This is possibly because the term ’event’ was not defined properly in the context and so users may have been confused as to what this was referring to. A possible solution might be to ask the participants if the major events of the game match that of the story as this would allow us to determine which events the participants consider to be 'major'.

The next section describes the quality of the game.

Scores:

Run	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Score	4	4	5	4	4	5	4	4	4	4

Additional statistics:

Mean	4.2
Median	4
Mode	4
Maximum	5
Minimum	4
Range	1

Table 5.3: Results for the map portion of the evaluation.

5.3.4 Technical Soundness

The results for the technical soundness rating are shown in Table5.4. Here we notice that the results obtained are significantly low. Several users described the graphics as being average and nothing exceptional. However, one candidate mentioned that he/she would not purchase the game based on its technological level. This provides some insight into the lower scores as it shows that users may compare the game to recent commercial titles. However, this is not surprising as the Glest engine was released in the year 2004. As mentioned by on of the participants, an advantage of having an older engine is that it will create games which are playable on older machines.

In the next section we discuss the results obtained from the interface and control scheme evaluation.

Scores:

Run	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Score	3	4	4	3	3	4	3	3	3	4

Additional statistics:

Mean	3.4
Median	3
Mode	3
Maximum	4
Minimum	3
Range	1

Table 5.4: Results for the map portion of the evaluation.

5.3.5 Interface and Control Scheme

The results for the interface and control scheme rating are shown in Table5.5. Interestingly only one test resulted in a score other than 4. The candidate from Test 1 felt that the control scheme was slightly difficult to learn. This is probably due to a lack of experience with strategy games. We draw this conclusion because several participants stated that control scheme followed the standard employed by other games. The next section discusses any additional comments which users provided.

Scores:

Run	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Score	3	4	4	4	4	4	4	4	4	4

Additional statistics:

Mean	3.9
Median	4
Mode	4
Maximum	4
Minimum	3
Range	1

Table 5.5: Results for the map portion of the evaluation.

5.3.6 Additional Comments

Participants were asked to provide any ideas which they believe might improve the game. While most candidates did not make any specific suggestions, one participant mentioned that the environment felt static and should have been influenced by the story. This was supported by another candidate who stated that the game should have been more interactive and detailed. However, as mentioned previously, generation of additional content is not the primary concern of our research.

While the above tests yield promising results, particularly in the area of accuracy, we must consider that results may have been skewed by external factors. The next section discusses possible improvements to the experimental process.

5.4 Experimental concerns

In performing our experiments much was learned about the methodology required to evaluate a Text-to-Game system. This chapter describes possible solutions for those experimental issues which should be addressed in future work. In the first section we consider the dangers introduced by user testing. Secondly, we consider alternative approaches to evaluation of the Text-to-Game system.

5.4.1 User testing issues

We recognise that the results of the experiment are positive. However, dangers are introduced with user testing in the form of many external (uncontrollable) factors. These dangers may cause extreme results skewed by personal preference or feeling. For example, a participants like or dislike for strategy games may influence the score which they give.

In order to deal with this issue a control test can been introduced. This would allow us to eradicate those external factors which might cause issues as results obtained in the experiment would be relative to those obtained in the control if participants are kept constant.

An example of such a control test would be as follows:

1. Users read a story which has already been [manually] converted into a game
2. Users then play the existing [possibly commercial] game based.
3. Users complete the questionnaire
4. Users play the game generated by the Text-to-Game system
5. Users complete another questionnaire

This would allow us to look at the results relative to a control test. This eradicates any external factors as they should be common to both the control and experimental tests.

5.4.2 Alternative Evaluation Methods

Evaluation of the system was purely focused on the game. It would also be beneficial to attempt evaluating the system as a game development tool. This is possible if we use the concept of a control test again. If we measure the amount of time and effort required to create a game manually, we can then examine the amount of time required to recreate the game using the Text-to-Scene system. On comparing these 2 results we would be able to determine if the Text-to-Game system can increase the rate and effort required to produce a game.

5.5 Summary

Issues with determining the accuracy of the game were encountered as some participants felt that the story lacked detail for certain elements. Also, interpretation of the game elements varied between participants meaning that some aspects of the game were not recognised by the participants. However, scores were favorable and all participants agreed that the map, characters and events were accurately represented by the game. In terms of the quality of the game, users felt that the technical aspects of the game are good but lag behind more modern games. However, in conducting these experiments we learn that issues may arise if a control test is not performed. Also, the qualitative nature of user tests prevents us from obtaining a quantifiable result for representing the performance of the system. An issue which could be overcome by employing a different research methodology.

The next Chapter summarises the results and findings of our research.

Chapter 6

Conclusion

6.1 Summary

Research was started through a categorization of related work in which we define the link between Text-to-Scene systems and game development tools. Further examination of related work showed that stories or fiction texts can be uniquely identifiable by 3 main elements. These include the environment, the characters and the events. We then carry this idea across to aid the design of the system. This lead to the concepts of a map and game generator which together are responsible for creating the environment, the characters and events. Next, we describe the implementation of the system in terms of 3 main components, the game engine, the map generator and the game generator. Evaluation of the system is then then discussed as we consider a user testing based methodology for determining the quality of game generated by the system and how accurately they represent their textual counterparts. We then give a discussion of results and mention possible improvements to our experimentation methodology. The next section concludes our research by relating the work done to the initial problem which we set out to solve.

6.2 Conclusions

In concluding we revisit the problems which we set out to solve. In doing this research we have solved the achieved the following:

1. Created a method for accurately converting the environment described in a story into a virtual map.

2. Developed a system with the ability to recreate the characters described in a piece of text as players in a virtual world.
3. Create a system capable of replicating the events described in a story as achievable game goals.

By solving these problems we have managed to create a accurate, working Text-to-Scene system.

6.3 Contributions

In performing this research we have contributed the following to this field of research:

1. Provided a new method for automated game development.
2. Created a rapid prototyping tool for use in game development.
3. By creating a game development tool which relies solely on annotations we have managed to simplify the game development process and separate the concept/story of the game from the actual implementation.

Appendix A

User Evaluation

A.1 Story for user evaluation

Tramus had just arrived in The Valley. He had been called upon to defeat the behemoths which had been causing havoc there for some time now. Not having visited The Valley before Tramus was slightly unsure of his surroundings and so continued along the mountains until he came across a path. He then followed the path to the South until he encountered an old man who greeted him openly. ‘Hello stranger’ said the old man. ‘I am Wiseman Pat. I believe you are the one who is trying to defeat the behemoths to the west. Take 20 gold pieces to the blacksmith and he may be persuaded to upgrade your armor and sword. He can be found in the woods to the North-East. Also, I heard a rumor about a stash of gold situated just beyond the mountains to the East’ Tramus then thought to himself ‘there is no way to be sure if this old man is crazy.’ But Tramus knew that defeating the behemoths would not be an easy task and so he began venturing to the east in search of the gold. Eventually he came into a clearing beyond the mountains where Tramus was faced by a strange looking daemon. As he drew nearer he caught a glimpse of something shiny behind the daemon. It was a pile of gold not yet removed from its natural state. Suddenly the daemon lunged toward him. Tramus quickly drew his sword and managed to knock the daemon unconscious. ‘That was lucky’ Tramus thought to himself. ‘I should take this gold and find the blacksmith.’ So Tramus collected the gold and headed back the way he came. After some time he noticed a small path heading amongst the some trees. He decided to follow this path and was pleased to see that he had found the blacksmith. ‘Welcome. I see you have some gold traveler. I will use this to upgrade your sword and armor’. Once the blacksmith had finished Tramus felt a new confidence and decided that it was time to find the behemoths. Remembering what Wise-man Pat had said, he started his journey to the west in search of the creatures. After some time came across a sparse area which seemed eerily quiet. After taking a few more steps Tramus turned his head and finally caught sight of what it was that he was called to defeat. Realising that he was the best chance The Valley stood against these violent beasts, he drew his sword and charged. With his new gear Tramus felt powerful and was able to overcome the beasts with little effort.

A.2 Questionnaire

Each of the statements are followed by a scale representing a range of responses. Users were asked to tick the appropriate response for each question and provide feedback on their reasoning.

Question 1:

The game map accurately matched the description of the world in the story.

Strongly agree	Agree	Neutral	Disagree	Strongly disagree

Question 2:

The characters in the game accurately represented the characters in the story.

Strongly agree	Agree	Neutral	Disagree	Strongly disagree

Question 3:

The events described in the story are accurately recreated by the game.

Strongly agree	Agree	Neutral	Disagree	Strongly disagree

Question 4:

Please rate the game technologically. (graphics, sounds, effects, etc.)

5 (best)	4	3	2	1 (worst)

Question 5:

Please rate the interface and control scheme of the game.

5 (best)	4	3	2	1 (worst)

Question 6:

Are there any improvements to the game which you could recommend?

A.3 Results

	Question 1	Question 2	Question 3	Question 4	Question 5
	Accuracy of map	Accuracy of characters	Accuracy of events	Technological soundness	Control scheme and interface
Test 1	4	3	4	4	4
Test 2	5	5	5	4	4
Test 3	4	4	4	3	4
Test 4	4	4	4	3	3
Test 5	4	3	4	3	4
Test 6	5	5	5	4	4
Test 7	4	4	4	3	4
Test 8	5	4	4	3	4
Test 9	5	5	5	4	4
Test 10	5	5	4	4	4
Mean	4.5	4.1	4.2	3.4	3.9
Median	4.5	4	4	3	4
Mode	4	4	4	3	4
Maximum	5	5	5	4	4
Minimum	4	3	4	3	3
Range	1	2	1	1	1

Table A.1: Results from user tests

Bibliography

[1] Dan Adams. Ign: Warcraft iii: Reign of chaos review. http://pc.ign.com/articles/363/363926p2.html, July 2002.

[2] Ola Akerberg, Hans Svensson, Bastian Schulz, and Pierre Nugues. Carsim: An automatic 3d text-to-scene conversion system applied to road accident reports. In *Research Notes and Demonstrations Conference Companion, 10th Conference of the European Chapter of the Association of Computational Linguistics*, pages 191–194, Budapest, Hungary, April 12-1 2003. Association for Computational Linguistics. URL `citeseer.nj.nec.com/563862.html`.

[3] Bruce Alcock. A procedural, minimal input, natural terrain plug-in for Blender. Technical Report Honours Project Report, Virtual Reality Special Interest Group, Computer Science Department, Rhodes University, Grahamstown, South Africa, November 2007.

[4] Lawrence Argent, Bill Depper, Rafael Fajardo, Sarah Gjertson, Scott T. Leutenegger, Mario A. Lopez, and Jeff Rutenbeck. Building a game development program. *Computer*, 39(6):52–60, 2006. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/MC.2006.189.

[5] Russell Beale and Matthew Bond. What makes a good game? using reviews to inform design. In *HCI 2009 - 23rd Annual Conference on Human-Computer Interaction*, 2009.

[6] Bob Coyne and Richard Sproat. Wordseye: an automatic text-to-scene conversion system. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 487–496. ACM Press, 2001.

[7] Funda Durupinar, Umut Kahramankaptan, and Ilyas Cicekli. Intelligent indexing, querying and reconstruction of crime scene photographs. In *TAINN2004*, pages 297–306, 2004.

[8] David Finkel, Mark Claypool, Michael A. Gennert, Fred Bianchi, Dean ODonnell, and Patrick Quinn. Teaching game development: At the intersection of computer science and humanities & arts.

[9] Kevin Glass. *Automating the Conversion of Natural Language Fiction to Multi-Modal 3D Animated Virtual Environments.* PhD thesis, Department of Computer Science, Rhodes University, Grahamstown, South Africa, August 2008.

[10] Kevin Glass and Shaun Bangay. Evaluating parts-of-speech taggers for use in a text-to-scene conversion system. In Judith Bishop and Derrick Kourie, editors, *SAICSIT 2005 South African Institute of Computer Scientists and Information Technologists*, pages 20–28, White River, South Africa, September 2005.

[11] Kevin Glass and Shaun Bangay. A naive salience-based method for speaker identification in fiction books. In *PRASA 2007: Proceedings of the 18th Annual Symposium of the Pattern Recognition Association of South Africa*, pages 1–6, November 2007.

[12] Kevin Glass and Shaun Bangay. Constraint-based conversion of fiction text to a time-based graphical representation. In *SAICSIT '07: Proceedings of the 2007 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pages 19–28, New York, NY, USA, October 2007. ACM Press. Best paper award.

[13] Kevin Glass and Shaun Bangay. Automating the creation of 3D animation from annotated fiction text. In *Proceedings of the IADIS International Conference on Computer Graphics and Visualization 2008*, pages 3–10, July 2008.

[14] Richard Johansson, Anders Berglund, Magnus Danielsson, and Pierre Nugues. Automatic text-to-scene conversion in the traffic accident domain. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1073–1078, Edinburgh, Scotland, July 2005.

[15] Craig A. Lindley. *Story and Narrative Structures in Computer Games.* High Text, January 2005. ISBN 393326992X.

[16] S. Louchart, I. M. T. Swartjes, M. Kriegel, and R. S. Aylett. Purposeful authoring for emergent narrative. In *Proceedings of the First Joint International Conference on Interactive Digital Storytelling, Erfurt, Germany*, Berlin, September 2008. Springer Verlag.

[17] Sandy Louchart, Ruth Aylett, Michael Kriegel, Joï¿œo Dias, Rui Figueiredo, and Ana Paiva. Authoring emergent narrative-based games. *Journal of Game Development*, 3(1):19–37, March 2008.

[18] Minhua Eunice Ma. Confucius: An intelligent multimedia storytelling interpretation and presentation system. Technical report, School of Computing and Intelligent Systems, University of Ulster, Magee, September 2002. URL `url{http://www.infm.ulst.ac.uk/~paul/phd/ma1styrrpt.pdf}`.

[19] Pablo Moreno-Ger, José Luis Sierra, Iván Martínez-Ortiz, and Baltasar Fernández-Manjón. A documental approach to adventure game development. *Sci. Comput. Program.*, 67(1):3–31, 2007. ISSN 0167-6423. doi: http://dx.doi.org/10.1016/j.scico.2006.07.003.

[20] Mark J. Nelson and Michael Mateas. Towards automated game design. In *AI*IA '07: Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence on AI*IA 2007*, pages 626–637, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74781-9. doi: http://dx.doi.org/10.1007/978-3-540-74782-6_54.

[21] Joel Rasdall. Strategy-gaming: Warcraft iii: Reign of chaos review. http://www.strategy-gaming.com/reviews/warcraft_3/index.shtml, September 2002.

[22] Mark O. Riedl, Andrew Stern, and Don M. Dini. Mixing story and simulation in interactive narrative. In John E. Laird and Jonathan Schaeffer, editors, *AIIDE*, pages 149–150. The AAAI Press, 2006. ISBN 978-1-57735-235-8.

[23] Scott Schaefer and Joe Warren. Teaching computer game design and construction. Technical report, Worcester Polytechnic Institute, 2004.

[24] Alexis Sepchat, Nicolas Monmarché, Mohamed Slimane, and Dominique Archambault. Semi automatic generator of tactile video games for visually impaired children. In Klaus Miesenberger, Joachim Klaus, Wolfgang Zagler, and Arthur I. Karshmer, editors, *Proc. ICCHP 2006 (10th International Conference on Computers Helping People with Special Needs)*, volume 4061 of *LNCS*, pages 372–379, Linz, Austria, July 2006. Springer.

[25] Richard Sproat. Inferring the environment in a text-to-scene conversion system. In *K-CAP 2001: Proceedings of the international conference on Knowledge capture*, pages 147–154. ACM Press, 2001.

[26] Jan Svartvik. Computer-aided grammatical tagging of spoken english. In *Proceedings of the 8th conference on Computational linguistics*, pages 29–31, Morristown, NJ, USA, 1980. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/990174.990180.

[27] Julian Togelius and Jürgen Schmidhuber. An experiment in automatic game design. In *Proceedings of the 2008 IEEE Symposium on Computational Intelligence in Games CIG-2008*, 2008.

[28] Javier Torrente, Angel del Blanco, Guillermo Canizal, Pablo Moreno-Ger, and Baltasar Fernandez-Manjon. e-adventure3d: an open source authoring environment for 3d adventure games in education. In *ACE '08: Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pages 191–194, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-393-8. doi: http://doi.acm.org/10.1145/1501750.1501795.

[29] Annika Wolff, Paul Mulholland, Zdenek Zdráhal, and Richard W. Joiner. Scenedriver: An interactive narrative environment using content from an animated children's television series. In Stefan Göbel, Ulrike Spierling, Anja Hoffmann, Ido Iurgel, Oliver Schneider, Johanna Dechau, and Axel Feix, editors, *TIDSE*, volume 3105 of *Lecture Notes in Computer Science*, pages 213–218. Springer, 2004. ISBN 3-540-22283-9.

[30] Xiaojin Zhu, Andrew B. Goldberg, Mohamed Eldawy, Charles R. Dyer, and Bradley Strock. A text-to-picture synthesis system for augmenting communication. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 1590–1595. AAAI Press, 2007. ISBN 978-1-57735-323-2.