DISE: A Game Technology-based Digital Interactive Storytelling Framework

SIMON COOPER

A THESIS SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS OF

LIVERPOOL JOHN MOORES UNIVERSITY

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

August 2011

Supervisors:
Prof. Abdennour El Rhalibi
Prof. Madjid Merabti

THE FOLLOWING HAVE NOT BEEN COPIED ON INSTRUCTION FROM THE UNIVERSITY

ABSTRACT

This thesis details the design and implementation of an Interactive Storytelling Framework. Using software engineering methodology and framework development methods, we aim to design a full Interactive Storytelling system involving a story manager, a character engine, an action engine, a planner, a 3D game engine and a set of editors for story data, world environment modelling and real-time character animation. The framework is described in detail and specified to meet the requirement of bringing a more dynamic real-time interactive story experience to the medium of computer games. Its core concepts borrow from work done in the fields of narrative theory, software engineering, computer games technology, HCI, 3D character animation and artificial intelligence.

The contributions of our research and the novelties lie in the data design of the story which allows a modular approach to building reusable resources such as actions, objects, animated characters and whole story 'levels'; a switchable story planner and re-planning system implementation, allowing many planners, heuristics and schedulers that are compatible with PDDL (the "Planning Domain Definition Language") to be easily integrated with minor changes to the main classes; a 3D game engine and framework for web launched or in browser deployment of the finished product; and a user friendly story and world/environment editor; so story authors do not need advanced knowledge of coding PDDL syntax, games programming or 3D modelling to design and author a basic story. As far as we know our Interactive Storytelling Framework is the only one to include a full 3D cross-platform game engine, procedural and manual modelling tools, a story editor and customisable planner in one complete integrated solution.

The finished interactive storytelling applications are presented as computer games designed to be a real-time 3D first person experience, with the player as a main story character in a world where every context filtered action displayed is executable and the player's choices make a difference to the outcome of the story, whilst still allowing the authors high level constraints to progress the narrative along their desired path(s).

ACKNOWLEDGEMENTS

Special Thanks to:

Prof. Abdennour El Rhalibi – Primary Supervisor

Prof. Madjid Merabti – Secondary Supervisor

Chris Carter, Chris Dennett, Hakim Sabri – LJMU Games Lab

Ricardo Duarte – LJMU Research Student

Marc Price – BBC R&D

Jon Wetherall – Onteca, http://onteca.com/

...and to all of my family and friends for their support.

DECLARATION

All material contained within and presented for examination is my own work and no part has been written for me by any other person(s). Where text and images have been quoted or paraphrased from other work(s) published or unpublished, there is due acknowledgement and a fitting citation to the referenced work.

TABLE OF CONTENTS

RESEARCH PUBLICATIONS

Here is a list of papers I have published during my time as a research student.

Cooper, S., El Rhalibi, A., Merabti, M., & Price, M. (2008). Dynamic Interactive Storytelling for Computer Games Using AI Techniques. 6th International Conference in Computer Game Design and Technology (GDTW). Liverpool: LJMU.

El Rhalibi, A., Dennett, C., Merabti, M., Fergus, P., **Cooper, S.**, Ariff Sabri, M. & Price, M., 2008. Homura: A Step Further Toward 3D Java Game Development Support. In *ACM ACE 2008.*, 2008.

Dennett, C., El Rhalibi, A., Fergus, P., Merabti, M., **Cooper, S.**, Sabri, M.A., Carter, C. & Price, M., 2008. 3D Java Game Development with Homura. In *6th International Conference GDTW 2008.* Liverpool, 2008.

El Rhalibi, A., Dennett, C., Merabti, M., Fergus, P., **Cooper, S.**, Ariff Sabri, M. & Price, M., 2009. 3D Java Web-Based Games Development and Deployment. In *IEEE International Conference on Multimedia Computing and Systems 2009.* Ouarzazate, Morocco, 2009.

El Rhalibi, A., Dennett, C., Merabti, M., Fergus, P., **Cooper, S.**, Ariff Sabri, M., et al. (2009). 3D Java Web-Based Games Development and Deployment. *IEEE International Conference on Multimedia Computing and Systems 2009, Volume: 2, Issue: 3-4*, pp. 553 - 559. Ouarzazate, Morocco.

Fergus, P., Kifayat, K., **Cooper, S.**, Merabti, M., El Rhalibi, A. "A Framework for Physical Health Improvement using Wireless Sensor Networks and Gaming", ICST/IEEE International Workshop on Technologies to Counter Cognitive Decline (TCCD), in conjunction with the 3rd International Conference on Pervasive Computing Technologies for Healthcare (Pervasive Health), 2009, City University London, UK, 31st March 2009.

Fergus, P., Kifayat, K., **Cooper, S.**, Merabti, M., El Rhalibi, A. "A Body Sensor Network and Gaming Platform for Dynamically Adapting Physiotherapy Treatments" at The Fifth IASTED European Conference on Internet and Multimedia Systems and Applications, Euro IMSA 2009, July 13-15, 2009, Cambridge, United Kingdom.

El Rhalibi, A., Carter, C., **Cooper, S.**, Merabti, M., Price, M. "Charisma: High Performance Web Based MPEG-4 Compliant Animation Framework", Journal ACM Computers in Entertainment, Vol. 8, Iss. 2., Article 8. Nov. 2010.

Cooper, S., El Rhalibi, A., Merabti, M., & Wetherall, J. (2010). Procedural Content Generation and Level Design for Computer Games. AISB 2010. Leicester.

Cooper, S., El Rhalibi, A., Merabti, M. & Price, M., 2010. DISE: The Digital Interactive Storytelling Engine. In *The 11th Annual Post Graduate Symposium in the Convergence of Telecommunications, Networking and Broadcasting (PGNET 2010)*. Liverpool, 2010.

Carter, C., **Cooper, S.**, El Rhalibi, A., Merabti, M., & Price, M. (2010). The Application of an MPEG-4 Compliant Animation to a Modern Games Engine and Animation Framework. Lecture Notes in Computer Science 2010, 6459/2010, pp. 326-338.

Cooper, S., El Rhalibi, A. & Merabti, M., 2011. DISE: A Game Technology-based Interactive Storytelling Framework. In *The 12th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNET 2011)*. Liverpool, 2011.

Duarte, R., El Rhalibi, A., Merabti, M., Carter, C. & **Cooper, S.**, 2011. An MPEG-4 Compliant Quadric-Based Surface Adaptative LOD. In *The 12th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNET 2011)*. Liverpool, UK, 2011.

TABLE OF FIGURES

> *"In computer games characters aren't presented, they are experienced (Noyle, 2006).*

1 INTRODUCTION

1.1 CONTEXT

As games become more complex their use as a storytelling medium is growing in importance and popularity. The interactive nature of games means that stories and characters can become more personal and involving.

Stories can be implemented in different ways: either linear, branching, parallel, or threaded. Games typically follow a linear storyline, where the events of the story are presented in a predefined sequence. It can be argued that making a player follow a defined story can diminish the interactivity level of a game; the player is, after all, following a pre-set path already laid out for him by the author. In order to still convey a story and allow the player to feel a high degree of interactivity, the concept of interactive or non-linear storytelling has to be introduced. Simply put, Interactive Storytelling presents the opportunity for players to have an input on what is happening in the game world they are placed in, to be the ones who dictate how certain events may come to pass.

1.2 GOALS & OBJECTIVES

Our goal is to design and evaluate a more complete Interactive Storytelling engine and framework/middleware, called 'The Digital Interactive Storytelling Engine' (DISE); which will consist of separate player and editor components for the creation and deployment of new story modules.

DISE will dynamically create interactive narratives which are focused on user's actions to create alternative storylines and points of dramatic tension. The engine is provided with knowledge of generic story actions met in many storytelling domains. The story designer is required to provide domain specific information, for example regarding characters and their: relationships, locations, actions and the key scenes and events that link them together. A planner creates sequences of actions that allow a non-player character to pursue high level goals. These goals can be defined at the start of a story, triggered as a response to certain events or 'injected' by the story manager that is pre-programmed by the story author. The user interacts with the story-world by moving

around in a first person viewpoint and making decisions on relevant actions by choosing the object to interact with and constructing action sentences from a set of available verbs by linking them with nouns or other verbs. Using this action the engine chooses and adapts new story lines according to the user's past behaviour, surrounding character's moods, environment, pre-determined event triggers and other variable factors.

Our research goals and objectives are as follows:

- Design and evaluate a more complete/comprehensive Interactive Storytelling (IS) engine/framework.
- Evaluate current approaches used in IS literature, consolidating the knowledge we have of these approaches and analyse the advantages and drawbacks of each system.
- Research suitable narrative theories and models to deconstruct reoccurring universal story elements.
- Research and create novel suitable computational models and their generality and usefulness in creating Interactive Storytelling systems and agent Artificial intelligence.
- Define and develop a novel interactive storytelling framework providing a novel approach to story design and play.
- Create user friendly editors so story authoring and level design can be done by someone with relatively little programming or graphics experience.
- Include animated characters to add a level of realism and immersion.
- Include procedurally generated content to speed up story world creation.
- Create a framework that allows a whole story to be saved in a pluggable package.
- Implement technical demos for each aspect of the framework.
- Develop suitable approaches to evaluate the framework.
- Disseminate the outcomes of our research.

1.3 RESEARCH METHODOLOGY

In order to achieve the research objectives outlined above, we intend to address the following questions:

- Which techniques and structures can be used in games to progress a narrative further, without breaking player immersion? What are the advantages and drawbacks of these techniques, and the related technological limitations and what improvements could be made to these techniques?
- How can a story be broken down into smaller subplots that can be experienced at the player's discretion, whilst keeping a strong plot structure and maintaining a consistent point-of-view, or create smooth transitions between them? What theoretical model should be used to link these small subplots together?
- Which AI planning methods used in current research are the most relevant or need to be revisited to fit in the framework and will these methods be suited to controlling and adapting the story in accordance to the player's actions? Can a novel AI approach or heuristic be applied?
- What Interactive Storytelling scenarios could be used to show flexibility and generality on dynamic and interesting story plots with multiple genres?
- What game technologies will be required for an Interactive Storytelling engine and what drawbacks will the game engine face when dealing with complex stories with large worlds and multiple characters?
- Can game assets/environments also be generated procedurally?
- Can characters be realistically animated and display their emotions?

In response to these questions the following research methodology was carried out for this project. The project consists of five stages applied through the following iterations:

1. **Literature Review and Critical Analysis** Review and consolidation of knowledge of existing Interactive Storytelling systems.
2. **Problem Analysis & Design** Evaluate narrative and computational models to build a comprehensive Interactive Storytelling framework.
3. **Framework Implementation** Implement the components of DISE using a fast prototyping approach
4. **Technical Demos** Implement a series of prototypes to show certain aspects of the framework as a proof of concept and evaluate its viability.
5. **Critical Assessment** Review the framework comparing it to the related work and stating its contributions and limitations.

1.4 CONTRIBUTIONS TO KNOWLEDGE

Previous Interactive Storytelling systems have architectures which are not capable of maintaining the change in dramatic tension of the narrative over a long time period or have a more limited branching narrative with a set number of endings. Other examples have little to no user interaction and can disembody players from an involving story experience by treating them as an omnipotent god like entity or by having limited graphical representations of the player and surrounding world.

This thesis makes the following contributions to knowledge. These are listed below categorised as either **primary contributions** or **secondary contributions** in respect to their importance in the field of Interactive Storytelling:

1.4.1 PRIMARY CONTRIBUTIONS

1. **Provide an up to date literature survey of existing Interactive Storytelling systems, both in research and commercial games.**

   Consolidating the knowledge of previous work in the field of Interactive Storytelling can help to identify techniques that provide good results and also to see the areas in Interactive Storytelling that have not be thoroughly explored yet and in which we could provide a novel perspective. Some of this literature survey was published in the following papers: (Cooper S. , El Rhalibi, Merabti, & Price, 2008), (Cooper S. , El Rhalibi, Merabti, & Price, DISE: The Digital Interactive Storytelling Engine, 2010) (Cooper, El Rhailbi, Merabti, & Price, 2010) and (Cooper, El Rhalibi, & Merabti, DISE: A Game Technology-based Interactive Storytelling Framework, 2011).

2. **Identify and create the key design features, most suitable narrative theories and computational models for use in an Interactive Storytelling framework.**

   By reviewing both the theory of how stories are formulated and computer science together, stories can be broken down into a formal language, represented on a computer and created procedurally using changing variables (Cooper, El Rhalibi, & Merabti, DISE: A Game Technology-based Interactive Storytelling Framework, 2011).

3. **Make a novel storytelling framework with full real-time interaction in a dynamic 3d world with the player taking the role of a main character in the story and looking through their eyes in a first-person perspective.**

   Examples of other Interactive Storytelling systems are provided in Chapter 3

'Related Work', but only a small portion of these include the player as a main character in the narrative, whilst running in real-time. In some systems the player is not actively involved in the narrative and is an omnipresent entity limited to passive indirect interactions with the storyworld (Cooper S. , El Rhalibi, Merabti, & Wetherall, Procedural Content Generation and Level Design for Computer Games, 2010) (Cooper, El Rhalibi, & Merabti, DISE: A Game Technology-based Interactive Storytelling Framework, 2011). Context filtering has been used in modern remakes of classic adventure games to improve the user's flow of interaction. By only showing possible actions to the player DISE will also remove the jarring experience of choosing or inputting an action only to find it can't be done or provides no response in the system.

4. **Create useable editors to manage story data and describe characters.**

   New stories are complicated to design without an editor and would require a huge alteration to the systems code base. Using a separate file format to hold story data independently from the engine and including GUI based editing tool will lower the barrier to entry for new story authors and means the source code of the main system does not need to be distributed.

5. **Create useable editors for 3d word/level design/editing and game assets.**

   An easy to use editor is needed to allow Story Authors to build their environments quickly and also load in and position 3d model files to represent complicated object such as props and scenery.

6. **Development of integrated solution for DISE Framework.**

   The aforementioned systems for storytelling, games, planning and editing need to be integrated into a comprehensive framework, which provides a supportive pipeline for creating and experiencing interactive stories, using the Homura Game Engine (Dennett, et al., 2008).

7. **Evaluation of DISE Framework**

   The DISE framework has to be evaluated with experiments that provide evidence in support of our thesis and emphasise either the proof-of-concept (demonstrating the viability of a method/technique) or efficiency (demonstrating that a method/technique provides better performance than those that exist), depending on the systems role in the overall framework.

8. **Dissemination of the findings**

The outcomes of DISE have been disseminated via a number of papers and our work has been presented at international conferences. These include (Cooper S. , El Rhalibi, Merabti, & Price, 2008), (Cooper, El Rhailbi, Merabti, & Price, 2010), (Cooper, El Rhalibi, & Merabti, DISE: A Game Technology-based Interactive Storytelling Framework, 2011) and (Cooper S. , El Rhalibi, Merabti, & Price, DISE: The Digital Interactive Storytelling Engine, 2010).

1.4.2 SECONDARY CONTRIBUTIONS

1. **Create a real-time facial animation system for storytelling with characters.**

   The Charisma system, we have developed, can be integrated into DISE to allow key-frame editing and real-time play back of MPEG-4 compliant facial animation synced with audio dialogue. This can be used to introduce a story or convey a particular message in a scene. Our work on the Charisma System is described in full in the publications "The Application of an MPEG-4 Compliant Animation to a Modern Games Engine and Animation Framework" (Carter, Cooper, El Rhalibi, Merabti, & Price, 2010) and "An MPEG-4 Compliant Quadric-Based Surface Adaptive LOD" (Duarte, El Rhalibi, Merabti, Carter, & Cooper, 2011).

2. **Procedurally create content, including art assets and level design from high level constraints controlled by the story author.**

   Procedural methods show great potential but are an underused solution to manual content creation. Limitations to these methods include the lack of control of the output due to its random nature and the absence of integrated solutions, although more recent publications increasingly address these issues, they are not usually incorporated into Interactive Storytelling Systems. Our research on procedural content was presented at the AISB in the following paper (Cooper S. , El Rhalibi, Merabti, & Wetherall, Procedural Content Generation and Level Design for Computer Games, 2010).

3. **Possibility for multiplatform and web deployment of the games/editors.**

   Using the Net Homura platform (Carter, Cooper, Dennett, & Sabri, 2008) we have created, it is possible to deploy DISE online from a Java WebStart or Applet directly from the most popular web browsers.

4. **Provides a pluggable planning system for further research and expandability.**

   By designing the planner to be self-contained with a Java class linking it to DISE, any new planners using the PDDL (Planning Domain Definition Language) format that support the correct requirements functions can be switched with the default planner. This means that comparison tests can be made and the speed and functionality of DISE can be increased in the future.

1.5 THESIS STRUCTURE

The rest of the thesis is structured in the following way:

Chapter 2 – Background: covering our research area, problem and motivation. In this section we present the background of our research area, which is Interactive Storytelling, demonstrate a wider appreciation of the subject (to give context), and provide our problem statement and motivations for this thesis.

Chapter 3 – Related work: a literature review of relevant work in the field. In this section we will survey and critically assess projects and publications related to the field of Interactive Storytelling, planning, and computer games technology and state their relation to our own work, along with their positive and negative aspects.

Chapter 4 – Interactive Storytelling Framework Analysis and Design: the analysis and design of our Digital Interactive Storytelling Engine (DISE) and its encompassing framework. The main focus of this section is to determine how Interactive Storytelling (IS) systems can be improved and look at what makes a story interesting and immersive. We will break down the problem, the deliverables and create a philosophy of our approach in picking the important ideas to build our Framework. It also contains our plan of attack to show we approached the problem systematically, whilst raising any design issues that could occur.

Chapter 5 – DISE Implementation: how the DISE framework design was finally implemented and made into prototype applications. In this section we will explain the inner systems and their data models in more detail and will also include some code examples.

Chapter 6 – DISE Framework Evaluation: what information we have gained from the benchmark and comparison tests. This section explains how we got our results and evaluates what they mean. We will describe the results of experiments that provide evidence in support of our thesis. We chose experiments that will emphasise either the proof-of-concept (demonstrating the viability of a method/technique) or efficiency (demonstrating that a method/technique provides better performance than those that exist).

Chapter 7 – Summary Conclusions: assessment of our hypothesis; and demonstration of its precision, thoroughness, contribution to knowledge and comparison with the closest rival; and also its limitations. This section is a summary of our thesis, what we learnt and how it was applied.

Chapter 8 – Further Work: the work can be carried out in the future to further enhance this project. This section contains information on missing functionality,

descriptions of variations, extensions, or other applications of our central idea along with the possibilities for future research.

> *"Read you a story? What fun would that be? I've got a better idea: let's tell a story together (Adam Cadre, Photopia, 1998).*

2 BACKGROUND

In this section we present the background of our research area, which is Interactive Storytelling, demonstrate a wider appreciation of the subject, and provide our problem statement and motivations for this thesis.

2.1 RESEARCH AREA

The research background contains a descriptive account of storytelling through history and compares its different forms and media of conveyance along with their popularity, reception and the importance of storytelling as a social and learning tool in human society. We examine the inclusion of stories in computer games over the last forty years and the new opportunities for story interaction presented by this relatively new format, whilst comparing techniques to more classically established media to evaluate the main similarities and differences to see what can be adapted to better suit the needs of an interactive medium. The following chapters focus on general storytelling in various media; then written storytelling and cinematic storytelling, with their similarities and differences to computer games; and finally how storytelling is currently used in games today.

2.1.1 STORYTELLING AND ITS MANY FORMS

Storytelling has been around as long as humanity has had language, maybe even earlier as primitive drawings found on cave walls suggest. "Our idea of self and our understanding of culture and the world is not only guided by the stories we hear, but shaped by the stories we tell" (Elrod, 2007). Children use storytelling in the act of play, to learn about the world and develop social skills. Mallan states "From a very early age their [children's] play takes the form of story; sometimes they term their story play 'pretend'... storytelling functions as a social, political, and educative activity" (Mallan, 2003).

Stories can be conveyed in different ways that use different techniques, styles and formality. Some most commonly used media are:

- **Oral often combined with expression and gestures** – can be someone's account of a real or fictional event or a radio drama, etc.

- **Theatre** – live oral & visual conveyance of a story by one or more actors/actresses.
- **Written** – can be a novel, play, script, journal, poem, choose-your-own adventure book, etc.
- **Illustrative** – paintings can tell a story, more recent forms are comics/graphic novels which are sometimes combined with written dialogue.
- **Cinema** – stems from theatre, pre-recorded film can combine audio, visual, and written storytelling.

Interactive Storytelling – much younger than the other forms, only developed over the last 40 years. This was originally in the form of branched written narrative with limited choices, but now implementing techniques from the aforementioned media – especially the visual aspect of cinema and the first-person narrative of written literature. Now unique in the aspect of interactivity where your actions can change the outcome of the story, with more depth and less limitations than interactive fiction/choose-your-own adventure books (Berlyn & Blank).

We will leave Interactive Storytelling for now and focus the next chapters on the written and cinema storytelling media mentioned above and how these have been adapted to the world of games.

2.1.2 COMPARING GAMES & WRITTEN STORYTELLING

Originally some early games borrowed heavily from written storytelling media such as novels. In 1973 Text Adventure/ Interactive Fiction (IF) games such as "Colossal Cave" were released and the genre gained huge popularity. "IF is a unique form of computer-based storytelling which places the player in the role of a character in a simulated world, and which is characterized by its reliance upon text as its primary means of output and by its use of a flexible natural-language parser for input" (Maher, 2006). These games were fully text based with no graphics and were similar in design to the choose-your-own adventure books. Since then as games grew richer in scope they still looked for guidance in written storytelling as it has been around for centuries and focuses on a first person viewpoint (or what's called a tight third-person written account), which is highly relevant and well suited to games where the player takes on the role of the lead character (Noyle, 2006). Interactive Fiction games eventually spawned Graphic Adventure games, which replaced some of the text with a point-and-click user interface input method and animated 2D (and later 3D) graphics. Some of the most popular Adventure games in the late 80's and early 90's were from the Lucas Arts Entertainment team and used a custom scripting language called SCUMM. Some popular SCUMM titles include Maniac Mansion, Day of the Tentacle, Sam & Max Hit the Road, the Indiana Jones Series, and the Monkey Island

Series.

2.1.3 COMPARING GAMES & CINEMA

Cinema has now been around for over 100 years and is said by Nelmes (Nelmes, 2003) to fall into three categories:

- **Classical narrative** – Evolved and sustained since 1920's Hollywood movement.
- **Anti-classical** – European model (also known as counter-cinema or art Cinema).
- **Avant-garde** – deliberately experimenting with many forms and construction of cinema to evolve a new style.

"Classical narrative is based on a set of simple principles. There is usually a central protagonist – the hero – who has a particular goal" (Nelmes, 2003). The story's hero engages with romantic, problematic or antagonistic encounters which prevent them from reaching their goal in a recognisable structure of causes and effects. The 'beginning', 'middle' and 'end' structure is used to establish the narrative, extend and complicate the story and lastly to resolve an outcome (usually a happy ending variation with goals completed and lessons or moral values learnt). This structure is similar to the Monomyth/Hero's Story mentioned in '2.1.1 STORYTELLING AND ITS MANY FORMS'. Cinema can create a powerful suspension of disbelief and evoke a wide variety of emotions in its viewers. Computer games have been compared to films in many respects as they have matured in graphical capabilities over recent years, now using three dimensional perspectives and cinematographic techniques to portray long narratives and characterisation.

The key difference is that classic cinema is a passive experience with the viewer taken along a static path consisting of a linear or clearly sequential and predetermined beginning, middle and end set out by a writer/director. The interesting comparison comes when looking at anti-classical or avant-garde cinema vs. computer games where "our role in this process is not merely a passive one" (Nelmes, 2003), but participants are asked to have a more active involvement, making sense of the individual scenes or trying to predict the path of the story. A more challenging narrative is portrayed using techniques that break the structure of classical cinema, such as: non-linear storylines; multiple protagonists; different realties (dreams, hallucinations, fantasies); mismatched sounds, edits, jump-cuts; and alternative endings, such as a sudden tragedy or plot twist. The problem lies that even these challenges can be overcome after multiple viewings as these twists will always play out in the same way and the characters will become predictable, which is one advantage of the deeper level of interactivity and direct involvement in

computer games, "In computer games characters aren't presented, they are experienced" (Noyle, 2006).

Although the level of interaction is superior, the genre of computer games is still in its infancy. Many modern games borrow techniques from classical and anti-classical cinema, but due to the different nature of the media story authors must use caution. Noyle talks about the errors in modern games that neglect "point-of-view", which can transpose emotional impact onto a characters action by drawing on their past experiences and current feelings towards what is happening. Another common error is when the point-of-view is suddenly changed too often and without transitional warning, for example from first to third person, or from one character to another, which can be a jarring experience for a player.

These errors can be avoided in cinema using traditional techniques, which were also previously used for a long time in theatre (where several characters may be played by the same actor), or even techniques used in novel writing. Games on the other hand require more thought as the player also does not want to be led by the hand, but to control their own experience.

This means the elements of storytelling that will be presented to the player must be discreet and follow a fairly consistent point-of-view, mainly the first person narrative. Instead of being told, "he climbed the fort wall", players want to live out the action first-hand ("I climbed the fort wall"). Emotion can also be drawn from the details surrounding the experience, for example: What was the wall like? How difficult was it to climb? What lies at the top?

Games such as The Legend of Zelda and Half-Life implement this first person narrative by keeping the lead character silent, allowing the player to project their own personality onto the protagonist and assume his/her identity (Bassos, 2004) (Bicknell).

2.1.4 STORYTELLING IN GAMES

Initially stories and games have different demands and constrains. Figure 1 below, as detailed in Murray's book (Murray, 2004), plots "game" against "story" as the X and Y axes of a graph respectively. The top right corner shows the area where these two meet and gives the game Myst as an example of a game with a story, whereas the bottom right contains the TV quiz show Mastermind which is purely a game show. In the past games have focused mainly on one of the story and game goals, falling closer to one axis than the other, with classic games like Super Mario Bros. only having the most basic story and focusing purely on skill and action gameplay, and text adventure games with a more detailed story, but limited game elements. The main difference between stories and games

according to Costikyan are "Stories are linear, though they can leap about temporally; they are experienced the same way every time. Games are non-linear, though they are experienced over time; game sessions are different each time" (Costikyan, 2005).

Figure 1: Story vs. Game (Murray, 2004)

Murray states, "The stories we tell reflect and determine how we think about ourselves and one another. A new medium of expression allows us to tell stories we could not tell before, to retell the age-old stories in new ways, to imagine ourselves as creatures of a parameterized world of multiple possibilities, to understand ourselves as authors of rules systems which drive behaviour and shape our possibilities" (Murray, 2004).

The new medium referred to is that of computer games. Since 2004 when this statement was written, that medium has advanced technologically allowing rule based systems to be more complicated than ever, with the latest computers and games consoles having multiple threads and processors running in parallel. With this extra power and greater storage capacity games are becoming bigger and more epic projects with sprawling worlds, an array of characters and a more complex narrative.

Fairclough believes that "The opinion that stories and games are fundamentally incompatible is out of date, after the success of many games that blend the two to create something that is different to both concepts. Half Life 2, Fable, Doom 3, and more games are coming out that will set new standards in story-based game A.I." (Fairclough C. , 2004)

According to Spector (Spector, 2007) the features common to all good stories are change, pacing, compelling characters and subtext.

- Player's actions "must change" themselves, the game world and the characters they meet.
- The pacing must vary to hold interest, with build-ups at the beginning and the end.
- Characters have to be interesting with their own personalities, goals, beliefs, needs and desires.
- The story should have a main plot subtext (content understood by the observer but not announced explicitly by the characters) as well as the smaller obvious sub-plots that make up the game.

Spector continues further by explaining five ways to approach storytelling in games: **rollercoaster, retold, sandbox, shared-authorship** and **procedurally generated** (Spector, 2007).

- **Rollercoaster** storytelling uses a pre-determined narrative, which players traverse in a linear manner with no significant options or unique experiences.
- **Retold** stories are abstract games with no story at all other than individual accounts of the game experience.
- **Sandbox** storytelling allows the player to interact with toys and create their own story through the choices that they make.
- **Shared Authorship** gives the player some freedom in a similar way to sandbox storytelling, but also has pre-determined goals as in rollercoaster. It is up to the player to decide the order they achieve the goals or to ignore them altogether.
- **Procedural**/Interactive Storytelling.

The last one, procedural (or interactive) storytelling, is the most interesting and is the focus of this research. It aims to give players more freedom to explore and develop relationships with the game characters and world through their choices. We will explain this technique in much more depth throughout the thesis, starting with the next chapter.

2.1.5 WHAT IS INTERACTIVE STORYTELLING?

According to Crawford Interactive Storytelling is "A form of interactive entertainment in which the player plays the role of the protagonist in a dramatically rich environment" (Crawford, Chris Crawford on Interactive Storytelling, 2004).

It aims to give players more freedom to explore and develop relationships with the game characters and world through their choices. Barber writes "An interactive narrative is a game world in which the user-controlled character(s) can physically and mentally

interact with (perceived) total freedom while experiencing a dramatically interesting story which is fundamentally different on every play – dependent on the user's actions. The user will be able to act as and when they desire, in ways which will have a perceivable long and short term effect on the story. They will be emotionally involved in this world, and will thus greater appreciate the compelling storyline which results" (Barber, Interactive narrative, 2008).

The main difference between passive and interactive stories is this level of user participation and involvement with the lead character.

The IGDA article on the "Foundations of Interactive Storytelling" (IGDA, 2001) defines Interactive Storytelling as "Any game featuring both characters and a story in which one or more narrative aspects changes interactively". They write that the possible narrative aspects that could be made interactive include:

- **Plot** – The most obvious route to Interactive Storytelling is by creating a plot that varies in response to the player's actions. Perfect Entertainment's 'Discworld Noir' and, to a much lesser extent, Konami's 'Metal Gear Solid' are examples of this.
- **Character Attitudes & Personality** – A more satisfying way to approach interactive stories is to look at ways that the player's actions might affect the attitudes of characters in the game world. We will look at several ways of approaching this later in this thesis.
- **Theme** – Although strictly hypothetical at this time, it is possible to conceive of a game in which the theme varies interactively. For example, imagine a game in which the story elements are mediated by the game's story engine in relation to that which the player has paid the most attention. If the player spent considerable time talking to a romantic interest, the theme of the story becomes biased towards a romantic element; if they focused on violent activity, the thematic details might evolve around an exploration of violence. Mastering dynamic plots and character attitudes will almost certainly be a prerequisite to exploring interactive themes.

2.1.6 POPULARITY OF GAMES VS OTHER MEDIA

We mentioned above that although computer games allow greater interactivity due to the nature of the medium, in the past they were not seen as a mainstream entertainment format; so would it be wise to choose this method of story delivery?

Computer games have grown in popularity in the last twenty years and are now reaching the mainstream market and even outperforming other more traditional media.

Mike Griffith, chief of Activision talked about US market statistics which showed that between 2003 and 2007 sales of movie tickets fell by 6%; the number of hours of TV watched dropped by 6%, sales of recorded music slumped 12% and purchases of DVDs remained flat, whilst over the same four-year period the video game industry grew by 40%. He stated "Video games are poised to eclipse all other forms of entertainment in the decade ahead" (BBC News, 2009).

2.2 AIM AND OBJECTIVES

Storytelling is an important part of human society, it is a social, political, and educative activity that broadens our idea of self and our understanding of the world and is also a popular form of entertainment and escapism.

The art of storytelling has been developed over many years and spans every form of media. Computer games are a relatively new format only gaining popularity in the last 20 years and sometimes said to be in their infancy and not showing as much depth and maturity as books and films. We will borrow some of the techniques mentioned previously from more firmly established media for both the narrative and visual aspects of games, but as they are unique in their level of interactivity we need to investigate new ways to generate stories procedurally that react to a players actions. To create a dynamic plot and interesting characters we will look at what has already been achieved in the field of Interactive Storytelling and evaluate some techniques used in computer science and AI research that could help us create complex interactions and dramatic events.

The objectives of this research thesis are to investigate the exciting field of Interactive Storytelling and its application to computer games. We will examine the techniques used in computer games to progress a narrative further, such as surroundings, events/interactions, point of view, music, characters, and evidential or personal items such as diaries or recordings, plot pacing and subtext.

Many approaches have been developed recently, as presented in '3 RELATED WORK'. These approaches are mostly based on rules, and planning techniques. Many others that use AI techniques based on heuristics, meta-heuristics, emotional models, norms group dynamic and agent systems are available and could be interesting areas to develop and contribute to in the perspective of Storytelling. Some of these techniques have been explored and used in similar context by our research team (El Rhalibi, Baker, & Merabti, Emotional agent model and architecture for NPCs group control and interaction to facilitate leadership roles in computer entertainment, 2005) (Carter, Cooper, El Rhalibi, Merabti, & Price, 2010). The most relevant of these will be further developed to provide a new narrative framework for emergent digital storytelling. Once the techniques have been

investigated and evaluated with appropriate games scenarios, a Storytelling game engine will be developed and evaluated in terms of flexibility, usability, and genericity in providing dynamic search approach in the development of a story.

Despite the growing interest in Interactive Storytelling, there have been only a small number of implemented demonstrators and few have attempted at developing a re-usable Interactive Storytelling technology. In this research we will propose such an Interactive Storytelling engine, which will be the result of a wide investigation on AI techniques not used in previous research. The system will be based on the Homura game engine for its visualisation component, while the narrative generation component will implement different model based AI, using a combination of rules, constraints, objectives, triggers, states, communication architectures and emotions.

The engine will use suitable AI techniques such as Planning and include systems based on heuristics to provide a new narrative framework for emergent digital storytelling. It will deal with interaction, non-player-character groups, dialogue and notes/messages found in the game world.

In the next chapter we propose to review and consolidate the related work.

> *"The player also does not want to be led by the hand, but to control their own experience.*

3 RELATED WORK

In this section we will survey and critically assess projects and publications related to the field of Interactive Storytelling, planning, and computer games technology and state their relation to our own work, along with their positive and negative aspects.

The goals of these research projects are to demonstrate the feasibility of certain theoretical and computational models and formalisms, "accordingly these efforts must be appraised not as working technologies but as speculative exploration of interesting concepts" (Crawford, Chris Crawford on Interactive Storytelling, 2004).

In the last two sections we analysed the importance of narrative theories and computational models for planning and scheduling. In the design of our Interactive Storytelling strategy, the former contains the most popular narrative theories and examines how a story can be formalised and deconstructed into its core components and the latter will look at computational planning models that can be used to control the story elements or characters and their AI to systematically generate narrative elements.

3.1 EXISTING INTERACTIVE STORYTELLING SYSTEMS & RESEARCH

This section surveys existing storytelling engines and popular algorithms or techniques, listed alphabetically, that may be useful for the development of our framework using a novel approach. We also highlight the key features and technologies that define the system or technique in each case, for example: Games Master, Directors, Goal Net, Fuzzy Cognitive Maps, Heuristic Planning, Hierarchical Task Network (HTN) Planning, Dilemma Based Logic, Case Based Reasoning, Genetic Algorithms, A Behaviour Language (ABL), User Interfaces, Alternate Reality, etc. For each project we have also included cited references to the appropriate papers and an easy to follow link to download the software, view a video, or find further information if these resources are currently available. Each sub-section is presented in the following format: **Title:** the name of the system or popular Interactive Storytelling technique being surveyed, **Key Features:** the system's key design characteristics, **Web/Download Link:** the full software download, video only or general

project information page, **Technical Synopsis:** includes the relevant figures, supporting theories and computational algorithms and finally the **Review:** the benefits and drawbacks of the system in relation to what we are aiming to achieve. After the review we will compare and cross-reference the following features and aspects of each system:

- **Narrative Theory** – does it follow a specific narrative theory e.g. five-act model or Aristotle's definition of dramatic actions?
- **Computational Models** – what computational model is used e.g. HTN planning.
- **Graphics** – how is the story displayed to the user, e.g. 2d, 3d, text or menu based GUI?
- **Audience** – who is the target audience for this system and what is their level of programming knowledge?
- **Scope** – what is the scope/aim/goal of the system and does it fulfil it?
- **Genre** – are stories told with this system limited to a specific genre, e.g. fairy tales/soap operas/etc.?
- **User Roles** – what part does the user play and what level of interactivity do they have?
- **Extendibility** – can any parts of the system be reused or expanded on and how easy is it to create new stories?

3.1.1 ALTERNATE REALITY STORY GAMES

Key Features:

- Characters controlled by actors & story designers, mainstream media makes stories feel real, players can interact directly with characters and each other.
- DL: http://wetellstories.co.uk/

The publisher Penguin UK and alternate reality game (ARG) designers Six to Start have recently launched a digital writing project called "We Tell Stories", challenging some of its top authors to create new forms of short story designed especially for the internet using games, blogs and web tools, such as Google Maps (Figure 2) (Penguin Books Ltd, 2009). "The first of the six stories is Charles Cumming's The 21 Steps, based on John Buchan's classic thriller The 39 Steps. It uses Google Maps and Google Earth to follow the trail of a bewildered young Londoner who witnesses a murder and is forced to smuggle a mysterious liquid on to a plane" (Rickett, 2009).

Other ARG examples can use a combination of online social tools and communities in addition to general mainstream media that's encountered every day and real world locations/meetings to tell a story that the community can interact with, for example:

- TV, Radio, Billboard, Viral & Guerrilla Marketing Campaigns.
- Social Networks - Facebook, MySpace, Twitter, etc.
- Web Apps - Google Maps/Earth, YouTube, Flickr, etc.
- Websites, Blogs, Forums, Wiki's, Automated Text Messages, Recorded Answer phone Messages & Actors.
- Retail Starter Kits, Trading Cards, Physical Puzzles.
- Physical Events, Meetings and Markers.

Review:

Some create fictional characters using actors that have real profiles on the above community websites & media, which tell a story through their life actions and recordings. ARGs have been used to promote TV series and films as part of a viral marketing campaign. Although a good example of Interactive Storytelling it is difficult to generate procedurally or using bots.

Figure 2: We Tell Stories Google Maps Example (Penguin Books Ltd, 2009)

3.1.2 ACTAFFACT

Key Features:

- BDI (belief-desire-intention) type plan-based practical reasoning agent framework, based on OCC Model and JAM BDI architecture, Java, 2D graphical display using the Batik Toolkit.
- DL: http://bit.ly/dYwxRK

ActAffAct (Affective Acting: An Appraisal-Based Architecture for Agents As Actors) (Rank, Affective Acting: An Appraisal-based Architecture for Agents as Actors, 2004) (Rank & Petta, From ActAffAct to BehBehBeh: Increasing Affective Detail in a Story-World, 2007) is an appraisal-based research tool that programs characters using Ortony's later revision of the Ortony, Clore & Collins (OCC) Model of Emotion (Ortony, 2003). The OCC Model is represented in code using the JAM framework, a "BDI-theoretic (Belief-Desire-Intention) agent architecture based upon the Procedural Reasoning System (PRS)" (Huber, 2001). Emergent narrative is created by the synthetic actor's responses to their environment in resolution of conflict "The conflicts between the characters in a play and the emotions involved in resolving them are the constituents of a dramatic structure, a plot" (Rank, ActAffAct, 2004). The graphics engine is 2D using the Batik Toolkit to render simple SVG characters and scenery.

Review:

ActAffAct has some limitations as a story engine, as explained by Axelrad and Szilas "Since the system is intended neither as an authoring tool nor as a presentation system, the options to interact and alter behaviour are limited and quickly require direct modification of the programme base" (Axelrad & Szilas, 2010).

Figure 3: ActAffAct Story Viewer Screenshot (Rank, ActAffAct, 2004)

3.1.3 BOVARY

Key Features:

- 3D graphics using UT2003 Engine, STRIPS Heuristic Search-based Planning (HSP) System, based on Madame Bovary Novel, uses characters feelings & desires to generate narrative, natural language text input.
- DL (Video Only): http://bit.ly/fyUfSl

Bovary (Pizzi & Cavazza, 2007) is an interactive storytelling system based on the novel Madame Bovary and generates a narrative driven by the desires and feelings of characters rather than focusing mainly on their actions. Bovary draws inspiration from Brémond's narrative model described in 3.2.5 ROLES & PROCESSES. It uses a STRIPS based real-time RTA* Heuristic Search Planner (HSP) along with the UT2003 Engine to create a 3D scene with animated characters that execute a specific feeling from a precompiled database as their story actions (Figure 4). The characters can also have a general overall feeling 'state' (for example content, sad, etc.) generated by the search cost values. The user can interact with the characters in the scene by typing instructions in natural language text to direct their next actions or change an emotional state, but the user is not a direct character in the story.

Figure 4: Bovary Architecture (Pizzi & Cavazza, 2007)

Review:

The Bovary prototype seems useful for dramatising scenes where characters discuss their feelings or make decisions based on emotions and can create content that would be difficult to express with purely action based planners. On the other hand planning based systems can have a range in the quality of their output and the many dialogue options have to be created by the story programmer/author (Axelrad & Szilas, 2010). A further prototype system was also made in 2009 which extends Bovary by adding EmoVoice emotional speech input (Cavazza, Pizzi, Charles, Vogt, & Andre, 2009), but was not available for testing.

3.1.4 CAROSA: A TOOL FOR AUTHORING NPCS

Key Features:

- Scheduler, Action/Object Dictionary, backwards chaining, HiDAC + MACES Crowd Simulation, Ogre Game Engine.
- DL: http://bit.ly/gFtrlR

In (Allbeck, 2010), Allbeck introduces CAROSA (Crowds with Aleatoric, Reactive, Opportunistic, and Scheduled Actions), a framework that allows "functional crowds" of NPCs to be more easily authored by storytellers rather than demanding that authors have a high level of programming knowledge to meticulously hand script NPC behaviour. The framework uses a system of goals related to locations, objects and activities to give the NPCs an active purpose. Allbeck notes that virtual characters can help to drive a storyline and provide emotional elements, but need to behave appropriately.

Figure 5: CAROSA System Diagram (Allbeck, 2010)

Figure 5 shows the components that make up the CAROSA framework and includes a PAR (Parameterized Action Representation) system, a crowd simulator (via HiDAC + MACES), the visualizer/game engine (Ogre), an action and object repository (the Actionary), a Resource Manager, Agent Processor and a Scheduler. The Actionary contains a list of actions that can be carried out by an NPC with their definitions and preparatory specification parameters that are used with backward chaining.

Review:

CAROSA features some ideas that would transfer well into an Interactive Storytelling engine, such as: an action dictionary which creates story independent reusable building blocks, an action queue for each NPC, an authoring GUI and a Scheduler; but lacks scalability (the simulation can run up to 30 characters simultaneously), character personalities, character animation using inverse kinematics and object site labelling (for specific interaction animations) and a planning system to manage resources more efficiently using heuristic search.

3.1.5 DEATHKITCHEN

Key Features:

- Depth-bound planner, 3d visualisation using UT2003 Engine, 3rd person point and click interface.
- DL (Video Only): http://bit.ly/h2cpJT

DeathKitchen (Lugrin & Cavazza, 2006) is an Interactive Storytelling prototype made using the UT2003 Engine that uses a depth-bound planning system to allow inanimate environmental objects to make decisions on how to react to a player's actions. This allows the surrounding environment to act as a character in the story. The player interacts by clicking objects in the scene and selecting an available action for their 3rd person avatar to carry out.

Review:

In the Death Kitchen urban horror story demo the kitchen is a character trying to harm the player by creating various accidents. The user input method is direct and easy to use and the interactions with objects and their responses are generated by the planner which allows events to be tied together, but there is no authoring tool so programming all of the objects behaviours could be complex.

Figure 6: DeathKitchen System Overview (Lugrin & Cavazza, 2006)

3.1.6 DEFACTO

Key Features:

- Java & VRML based, 3d world, Aristotelian arc model with increased suspense, first order logic.
- DL: http://bit.ly/hqifH7

Defacto (Sgouros, 1999) is an IS system that uses an Aristotelian arc structure to unfold a Greek tragedy. The user moves about in a 3D-VRML scene and at key moments chooses from a set of multiple choice text options. Defacto's algorithm runs in two main stages, the first one generating the sequence of actions for the characters depending on their goals and the choices made by the user; and the second one evaluating interesting situations and story outcome according to a set of dramatic rules.

Review:

The system was fully implemented and "is highly generative, thus providing a potential solution to the core algorithmic issue of interactive storytelling (narrative paradox)" (Axelrad & Szilas, 2010), but its graphics and gameplay are now very dated; also the complicated algorithms and lack of authoring tools make it very hard for researchers to expand on.

Figure 7: Defacto Plot Manager Architecture (Sgouros, 1999)

3.1.7 DRAMACHINA

Key Features:

- Text-based interactive fiction editor independent of a specific IF architecture, XML Output.
- DL: http://www.irisa.fr/prive/donikian/Research/index.html

DraMachina is an interactive fiction narrative authoring tool. The editor is mainly text based, so that authors of classic fiction can write an interactive story along with character descriptions. Donikian and Portugal structure their data using a file directory model based on film and drama morphologies and is defined as follows (Donikian & Portugal, 2004):

- "**Authors-directory**: each author can enter his own reference."
- "**Narration-directory**: this directory includes acts, periods, dramatic actions and unit's description."
- "**Objects-directory**: description of objects important in the course of the story."
- "**Areas-directory**: description of locations of the story."
- "**Actors-directory**: this directory includes elements related to the description of characters, which is composed of their characteristics, psychology, actions they can perform, roles they can play and relationships between actors."
- "**Scenes-directory**: detailed description of scenes."
- "**Dialogs-directory**: dialog edition based on protodialog patterns."

Actors are assigned default values for their characteristics such as walk speed, talking/listening focus, available actions and roles. Their moods are determined by 'strokes', which have an impact value (positive/negative emotion strength relating to happiness or trauma) and the duration time that the emotion will last for.

Donikian and Portugal use multiple narrative theories in their structural design including Bremond's Roles and Processes[1] for the overarching concept, the Three-Act Paradigm's definition of acts and scenes; and the Ethical Dimension system of character morals/values.

[1] More information can be found in section 3.2.5 ROLES & PROCESSES.

Figure 8: DraMachina UI Showing Object Hierarchy (Donikian & Portugal, 2004)

Review:

DraMachina is a valuable tool for writers to author narrative concepts in structured XML, which can be interpreted by programmers, but is only in prototype form and would take time to extend and develop further; although some of the structural concepts (such as the definition of scenes, areas, actors and objects) could be borrowed and used in conjunction with other narrative theories and computational models.

3.1.8 DUNGEONS & DRAGONS AND THE GAMES MASTER

Key Features:

- Games Master controls & adapts story.
- DL: http://www.wizards.com/dnd/

One of the earliest games to use Interactive Storytelling was in fact a table top game Dungeons & Dragons, which was released in 1974 (IGDA, 2001). A player is appointed the role of games master (GM), whose job is to direct the players through the dungeon, whilst reacting to their actions, using the original story as a guide and filling in the blanks using their imagination and reasoning for a more entertaining experience. In computer games the computers rule systems, which have to decide how to respond to player actions, replace the GM. The rules aren't capable of responding to any action in the same way a human can, so the freedom in the number of actions that can be performed at a given time is significantly less (IGDA, 2001).

Tychsen, et al. (Tychsen, Hitchens, Brolund, & Kavakli, 2005) list five areas, for which the GM is responsible, regardless of the game platform; which the automated storytelling engine must cover as:

- **Narrative flow**: Creating the scenario; either the pre-planned plot or environment that the game takes place in, or creating the scenario on the fly changing the narrative by interacting with the players.
- **Rules**: Making sure players know and follow the game rules & mechanics, such as how the world operates and what interactions can take place between players and the environment.
- **Engagement**: Keep players motivated and interested, by providing a fun experience with a good difficulty curve.
- **Environment**: Providing a fictional setting for the game with interesting surroundings and engaging characters.
- **Virtual world**: unique to computer games, the GM must also act as the game engine providing on-the-fly updates of the game world and the active agents, as required by the actions of the players and the narrative development.

Review:

A classic problem with computer role play games (CRPGs) compared to the table top games like Dungeons & Dragons is that the communication between the players and the GM is limited through computer input and output, whereas in person players can use Speech, Emotion and Body Language to interact (Figure 9).

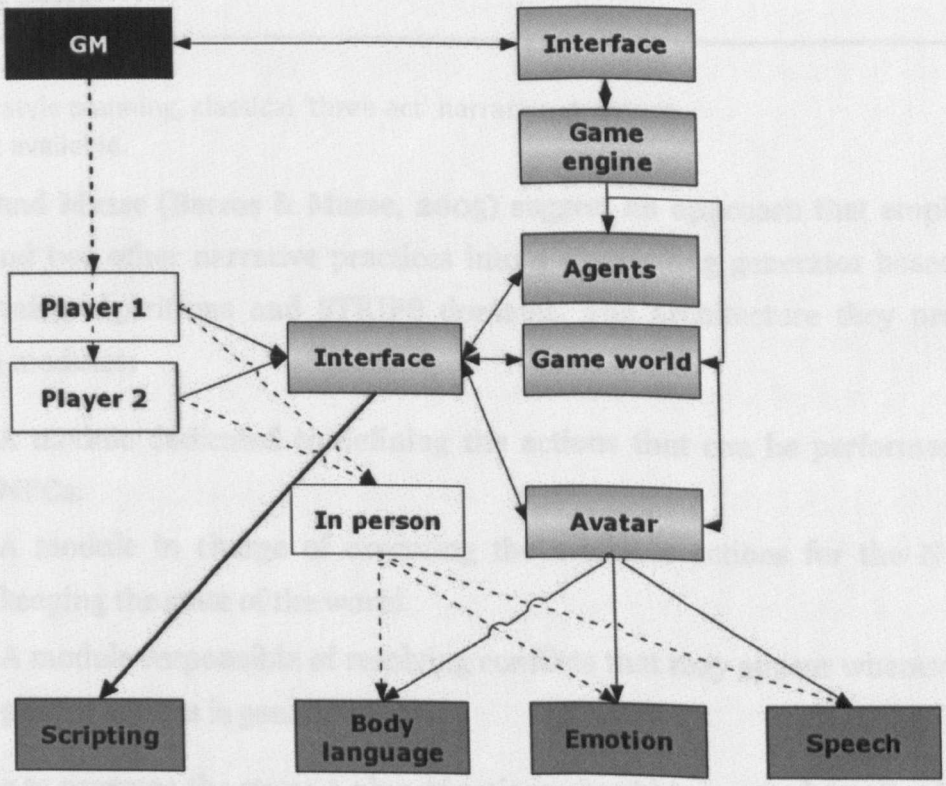


Figure 9: Communications Flowchart of a GM-Controlled Multi-player CRPG

3.1.9 FABULATOR

Key Features:

- STRIPS style planning, classical 'three act' narrative structure.
- DL: not available.

Barros and Musse (Barros & Musse, 2005) suggest an approach that employs this mechanism and two other narrative practices into a storytelling generator based on the usage of planning algorithms and STRIPS domains. The architecture they propose is divided into 3 modules:

- A module dedicated to defining the actions that can be performed by the NPCs.
- A module in charge of executing the available actions for the NPCs and keeping the state of the world.
- A module responsible of resolving conflicts that may appear whenever a new plan of actions is generated.

In order to progress the story, a plan of actions should be created for the characters to follow; each action is described in a language similar to STRIPS and is composed of a prerequisite, used to decide if the action is coherent given the current state of the world; and an effect, which indicates the repercussions of performing the action. For example, the action of giving a present to someone may have the prerequisites of having the present and being at the same location as the receiver, while the effects would be the receiver gaining an increased affection towards the gift giver. The implementation made by Barros and Musse "treats the planning problem as a state space search problem, and uses the A* algorithm to solve it" (Barros & Musse, 2005).

Barros and Musse incorporate into their model the following three narrative principles for creating interactive storylines:

- View story as a whole: By considering the overall storyline as a set of sequential plans, which in turn are sets of sequential actions, one can make sure that each plan will try to include the best actions to take in order to continue with the main plot; this kind of control will yield more coherent and believable stories.
- Three act storylines: Being the most common format for stories, it presents the opportunity to group together events that help develop the story: the first act introduces the overall problem, the second one helps increase tension and develop solutions and the third one presents the resolution of the story (Rollings & Adams, 2003) (Barros & Musse, 2005).
- Avoid narrative stalls: Being the focal point of the story, the player may sometimes decide to not follow the plan set in order to continue the story. To

avoid this, the actions of the NPCs can be prioritized, so that they can act and accomplish goals before the player. This will allow the story to be fluent even when the player gets stuck.

Review:

By combining the narrative techniques and the described architecture, Barros and Musse (Barros & Musse, 2005) were able to create interactive storylines that would adapt to the player's actions, even if those actions were not doing well in keeping the story fluent. One possible drawback might be the use of "predicate logic to represent the world state" (Barros & Musse, 2005) due the fact that it does not help us whilst dealing with the more complex aspects such as the "subtleties and nuances necessary for compelling stories" (Crawford, Chris Crawford on Interactive Storytelling, 2004); however, it can help us understand the problems and their solutions in an easier manner. Fabulator was extended by the authors to create a riddle based system using tension arcs and is explained further in 3.1.21 RIDDLE MASTER.

3.1.10 FAÇADE

Key Features:

- ABL (A Behaviour Language), NLU (Natural Language Understanding), Jess, real-time 3D (C++ OpenGL), Drama Manager, Story Beats.
- DL: http://www.interactivestory.net/

Mateas and Stern created an experiment in interactive drama called Façade (Mateas & Stern, 2003), where the player has to interact with two NPC's that are having marriage problems in their upscale apartment. The characters are programmed using A Behaviour Language (ABL). "ABL is a reactive planning language, based on the Oz Project language Hap, designed specifically for authoring believable agents - characters which express rich personality, and which, in [this] case, play roles in an interactive, dramatic story world." The ABL compiler is Java based, so could be considered for our project, which is based on a Java game engine. The story is controlled by a drama manager that breaks action into story beats (similar to small scenes), which trigger using preconditions and depend on the user's actions and current overall tension level to create an Aristotelian Arc style partially ordered drama. As a user navigates from beat to beat the characters are driven through different arguments and resolutions by their ABL code.

Review:

Façade is the only published and most complete interactive storytelling system with natural language processing, a 3d world, believable agents and the player controlling a main character, but wasn't designed to be modified so is a closed system with no authoring tools. Also the user's actions don't have a great impact on the future outcomes of the story.

Façade's novel features (Figure 10) include:

- The player's actions have a significant influence on what events occur, which are left out, and how the drama ends.
- Drama manager monitors the simulation and adds and retracts procedures (behaviours) and discourse contexts by which Grace and Trip (the NPC's) operate.
- Plot is made up of dynamically sequenced story beats.
- The multi-agent co-ordination is sequenced by these beats.
- Supports joint behaviours.

Figure 10: Façade Interactive Drama Architecture (Mateas & Stern, 2003)

3.1.11 FEARNOT

Key Features:

- Child centric, structure based on Forum Theatre.
- DL: http://bit.ly/gq3rOD

FearNot is a bullying simulator storytelling system for children created for the EU framework funded VICTEC project (ecirweb, 2006). The unscripted 3D characters use cognitive and emotional modelling to drive their behaviour and create an overall story from complex interactions without the need for a story manager. Story authors can assign characters goals and features and set which preconditions trigger an episode. The user watches a clip of a bullying scene and then talks to the victim using a text-chat window to advise them on what to do next (Figure 11). The next scene is then played out and can vary depending on the users talk with the victim.

Figure 11: FearNot! Victim Interaction (ecirweb, 2006)

FearNot's architecture uses the OCC Model for character/agent emotions in a similar manner to ActAffAct, but in this case it is paired with a unique double appraisal system that bases agent's decisions on their estimated effect on the other agent (Figure 12). This is split into a reactive level that uses emotional rules based on Elliot's Construal Theory that provides a quick reaction and a deliberative level which is slower but offers more "complex and rich behaviour" (Louchart, Aylett, Dias, & Paiva, 2006).

Figure 12: FearNot! Agent Architecture Diagram (Louchart, Aylett, Dias, & Paiva, 2006)

Review:

The FearNot system is novel in its ability to create unscripted autonomous characters using double appraisal and its use in schools as a bullying simulator/tutorial. Its limitations are in the way the user interacts with the story; players are invisible friends that can only interact with characters at certain points via messenger style text chat-bot, which can make the scenes appear scripted even though they are not.

3.1.12 GADIN

Key Features:

- Planner, Dilemma Based Narrative.
- DL: http://www-users.cs.york.ac.uk/~maria/gied/

Barber and Kudenko created an Interactive Storytelling system based on dilemmas called GADIN (Barber & Kudenko, Dynamic Generation of Dilemma-based Interactive Narratives, 2007). The system uses a story planner to create a plot with various dilemmas which ask for user input to resolve each one (Figure 13). The use of dilemmas is similar to the notion of conflicts in Aristotle's dramatic theory and screen writing (3-Act Paradigm and 5-Act Model). To create a GADIN story the following data is needed:

- Actions – allows characters to perform action or change their emotions and principles.
- Dilemmas – have to be initialised.
- Events – logged actions carried out by characters.
- Piece of Knowledge – a single fact, along with preconditions, reasons to share, how to know/un-know it.
- User Model – record of user's choices used to create new dilemmas.
- Story Length – finite or infinite.

Review:

The advantages of GADIN are its use of narrative dilemmas to extend planning systems and the soap opera like nature, which allows for on-going story generation with no definitive end. On the down side the user interface is 2d and completely GUI menu/text based.

Figure 13: GADIN Architecture (Barber & Kudenko, Generation of Dilemma-based Interactive Narratives with a Changeable Story Goal, 2008)

3.1.13 HEFTI

Key Features:

- HEFTI, Evolutionary/Genetic Fuzzy Algorithms, 2D Microsoft Agent, Integrated Authoring Environment.
- DL: not available.

Genetic algorithms have been used to solve many computer science problems as an adaptive heuristic search algorithm designed to find an exact or approximate solution, optimised over several generations of abstract populations in a computer simulation. Also known as evolutionary algorithms, they are designed to mimic evolutionary biology methods such as natural selection, crossover of genetic material, inheritance and mutation. These techniques give a diverse population of solutions, which are then evaluated, scored a fitness value, then re-selected for the desired number of generations, or until an appropriate fitness is reached. Ong, TeongJoo and Leggett, J. use this approach in their work to generate interactive narrative (Ong & Leggett, 2004). Their engine, the Hybrid Evolutionary-Fuzzy Time-based Interactive (HEFTI) Storytelling Engine uses genetic algorithms to recombine and evaluate story components generated from a set of story templates.

Review:

Figure 14 shows that "A gene is generated by constructing valid story element sets based on the current state of the story and the various conditions and rules imposed by the author. However, dependencies and ordering of various story elements dictate that the encoding process is strictly sequential since rules and story element selection will be resolved gene by gene" (Ong & Leggett, 2004). HEFTI has an integrated Authoring Environment (IAE) and a Graphical Object Library to create new stories with. One of the drawbacks of this system is that the IAE is based on the XML language, which is "program code for programmers, incomprehensible to the kind of creative talent needed for storytelling. So long as development environments look like this code example, only programmers will write storyworlds, and we'll continue to get the same old clichéd stories" (Crawford, Chris Crawford on Interactive Storytelling, 2004).

Figure 14: Encoding and Decoding of Story Components (Ong & Leggett, 2004)

3.1.14 IDTENSION

Key Features:

- Java based, centralised goals
- DL: http://bit.ly/ij1cfb

IDtension is a Java based Interactive Drama Engine by Dr Nicolas Szilas (Szilas, 2008). In the demo the user decides the actions that they would like the main character to perform using a text menu interface.

- Goals
- Tasks
- Segments
- Obstacles
- Values

The amount of actions a user can perform is much greater than some of the other systems mentioned in this study (up to 100).

Review:

Limitations – some of the actions get no response from certain characters and creating stories is complicated.

3.1.15 INTERACTIVE STORY ENGINE

Key Features:

- Case Based Planning, Story Director, Propp Morphology, MMO Client-Server Architecture, C++.
- DL: not available.

Fairclough, R. & Cunningham, P (Fairclough & Cunningham, 2003) investigated a multiplayer story engine that uses case based reasoning/planning to create a story adventure MMO game. "The system handles multiple users in a game world and directs the non-player characters therein to perform for the users parallel storylines, interweaving character roles in each story. The story is told through a 'narrative of actions' and automatically generated dialogue" (Fairclough & Cunningham, 2003).

An omnipotent director agent is used to control the story; by including a multiplayer aspect their director must be more capable and "assign story goals that are relevant to each player". Fairclough & Cunningham also base the story structure on Propp's analysis (Propp, 1977) and use functions that play on the hero & villain theme. Their system also includes interactive objects that can enable specific actions or transportation.

The story world is inhabited by computer controlled non-player characters that utilise a five part model:

- Low level control behaviours – e.g. path-finding.
- Social simulation – gossip system.
- Idle behaviours – such as patrol area.
- Targeted behaviour – goal search and execution.
- Attitudes – characters score other characters that they meet or hear about and record events for later gossip.

Figure 15: System Architecture for Multiplayer Stories (Fairclough & Cunningham, 2003)

The Interactive Story Engine has three demo games, an original story called "Bonji and the magic peanut", a story based on the film Star Wars: A New Hope and a third demo which includes multiplayer and more action objects.

Review:

The Interactive Story Engine allows the player to roam free around the 3d world and click on interactive objects to act on and change the path of the story. Using their Story Director (SD) agent and case-based planning different story structures can be generated "allowing the Story Author a higher level control of story events" (Fairclough & Cunningham, 2003). The multiplayer component and real-time play allows players to compete for NPCs loyalties, with persistent characters useful for creating a consistent online RPG adventure story.

3.1.16 LOGTELL

Key Features:

- Goal-inference planning, 3rd person viewpoint, 3D visualisation, based on Propp's morphology of the folktale.
- DL: http://bit.ly/tK7EnE (video only).

Ciarlini, M, et al. created a tool called LOGTELL to interactively generate and dramatize stories (Ciarlini, Pozzer, Furtado, & Feijó, 2005). Their system uses goal-inference rules, planning and user intervention to drive a 3D drama visualisation that can be used for entertainment or story creation within a specified genre.

Review:

Its main focus is on generating a large variety of coherent stories, rather than "creating an immersive experience in which the user takes part in the story as one of the characters". For this reason they use a third person perspective, with indirect/passive user interaction. LOGTELL borrows some story structure from Propp's work[2] on the topology of Russian folk tales (Propp, 1977) and "comprises a number of distinct modules to provide support for generation, interaction (management) and visualization of interactive plots", with the arrows representing the dataflow (Figure 16).

Figure 16: LOGTELL's Architecture (Ciarlini, Pozzer, Furtado, & Feijó, 2005)

[2] Described further in section 4.1.4 Propp's Morphology of The Folk Tale

3.1.17 SIFTABLES

Key Features:

- Unique hardware & user interface: a unit is combined with other units and a pc.
- LINK: https://www.sifteo.com/

Merrill & Kalanithi (Merrill & Kalanithi, 2008) created a physical object based children's storytelling game for learning and language development using the Siftables interaction platform. "Siftables are cookie-sized computers with motion sensing, neighbour detection, graphical display, and wireless communication. They act in concert to form a single interface: users physically manipulate them - piling, grouping, sorting - to interact with digital information and media. Siftables provides a new platform on which to implement tangible, visual and mobile applications" (Merrill & Kalanithi, 2008)(See Figure 17).

For the storytelling game several Siftables each depict an object or character which when moved around or connected together change the state of the story on the main view screen, which displays the whole scene. To bring a character on stage the user lifts the character's Siftable off the table so the motion sensor will detect the action and change the scene. When two character's Siftables are placed adjacently they will form a simple interaction, e.g. when the cat and dog Siftables were moved the scene changed to show "The Cat and Dog Say Hello!"

Review:

The object centric approach is easy to understand and the Siftable hardware is a natural way to interface with the story, but the story and interactions are limited to simple children's stories with only a few different combinations (around $\frac{n^2-n}{2}+n$, where n is number of Siftables). Siftables have recently been re-named to Sifteo and launched commercially as a trial run with 12 different games/apps and a more polished hardware design.

Figure 17: The Siftables Hardware (Merrill & Kalanithi, 2008)

3.1.18 SLEEP IS DEATH

Key Features:

- Two player only, collaborative storytelling experience, turn based and 2D graphics using STL.
- DL: http://www.sleepisdeath.net/

Sleep Is Death is a 2D storytelling game for two players by Jason Rohrer (Rohrer, 2010) that blurs the line between playing and creation of stories. One person fulfils the role of the 'player' with the other acting as the 'controller'. The player takes their turn exploring the screen; talking by typing text, which appears next to them in a speech bubble; and performing actions by typing the action and moving it with the mouse to point at its subject. Next control is passed to the controller, whose job is to react to the player and progress the story by manipulating the scene using the editor view. The default time for a turn is 30 seconds, but this can be altered in the settings for beginners and the turn can be ended early by clicking the advance button.

The editing view (Figure 18) contains tools for modifying the background, player object, general objects, sprites (a small image that can be used to build objects), creating music and adding text boxes/speech bubbles. Also included is a library of rooms, tiles, objects, sprites and whole scenes. If the controller creates new assets they can be saved to their library for later re-use, and also saved as resource 'PAK' files, which can be shared online.

Figure 18: Sleep Is Death Editor View (Rohrer, 2010)

Sleep Is Death archives the completed story as a website folder containing a slideshow of images showing the screen state that was submitted for each step of the story

which can be easily shared and viewed later.

Review:

Sleep Is Death (SID) is a successfully completed storytelling game, with a large community of storytellers, players and object/content builders. The graphics are simple 2d sprite based and static, but using clever art techniques a variety of view-points can be used. By letting a human control the editing and sending the modified world state across in a manner similar to the Games Master (GM) technique, any situation can be handled and depending on their imagination and authoring skills any genre and topic can be used. The negative points of SID are apparent when looking at our specific design goals, as it takes an alternative approach to Interactive Storytelling and is a completely differently designed system to the one we aim to create. This is not an entirely bad thing, but some negative points are the lack of true 3D graphics, the absence of any animation system (which means that characters jump around the screen when their position is changed by the GM), the difficulty of mastering how to use the editor fast enough to be prepared for any situation under the turn clock's demand, SID is turn based rather than real-time (which means waiting times can break the story's pacing and immersion), and the lack of a mode where the computer AI can control the game for a single player experience.

Figure 19: Shannon vs. Jason: Are We Home? (Rohrer, 2010)

3.1.19 S-MADE

Key Features:

- Multi Agent Development Environment, Goal Net, Fuzzy Cognitive Goal Maps, Drama Manager, Active Worlds 3-D virtual environment powered by Renderware.
- DL: not available.

Another technique generating Interactive Storytelling is described by Cai et al (Cai, Miao, Tan, & Shen, 2006). In this approach, a tool S-MADE, (Multi-Agent Development Environment for Storytelling) based on Goal Net is used to plan the story and Fuzzy Cognitive Maps are employed to analyse the user inputs and decide which path should be followed; they were selected due the fact that they act as "collection of the rules such that it not only concerns the relationships between the causes and effects, but also considers their relationships among the causes" (Cai, Miao, Tan, & Shen, 2006). Similar to the work described by Champagnat, Estraillier and Prigent, this approach implements an agent, modelled using the Goal Net tool, which will be in charge of presenting the story in accordance to the user actions (Champagnat, Estraillier, & Prigent, 2006) (Cai, Miao, Tan, & Shen, 2006). In order to accomplish this, a controlling agent keeps information related to the states required to achieve a goal, and the relationships or transitions that connect those states (Cai, Miao, Tan, & Shen, 2006).

Review:

Goal Net's ability to break goals down into simpler states make it useful for Interactive Storytelling (Cai, Miao, Tan, & Shen, 2006) (Shen, Miao, Tao, & Gay, 2005). Scenes can be divided into smaller segments creating multiple routes to the main plot subtext. It also has four different ways of connecting the small segments states (sequence, concurrency, choice and synchronization) allowing for flexible and more interesting transitions between the different states, which leads to the creation of more complex interlinking stories. Also important to note is that according to Cai et al, using Goal Net as planning tool instead of Hierarchical Task Network yields better results in terms of Interactive Storytelling, since it provides the ability to select scenes at real-time, according to the current context and user input (Cai, Miao, Tan, & Shen, 2006).

The engine created by Cai et al consists of a knowledge database, where the scenes and their relations are stored, the fuzzy cognitive goal net engine, a container for the agents to be used, known as the Drama Manager, and a Multi Agent Development Environment (MADE) platform to implement the agents' system; using fuzzy cognitive maps, the engine will decide, at runtime, which scenes from the knowledge database should be loaded into the agents in order to better suit the path selected by the user (Cai, Miao, Tan, & Shen, 2006) (Shen, Miao, Tao, & Gay, 2005).

Three main advantages can be distinguished from employing this method to create Interactive Storytelling (Cai, Miao, Tan, & Shen, 2006) (Shen, Miao, Tao, & Gay, 2005):

- All events are simplified into less complex scenes
- The engine can react not only to the user inputs but also to the actual state of the game, and look for the best path.
- Since all the events are loaded into the engine in real time, its performance is increased.

3.1.20 STORYTRON

Key Features:

- Uses verbs & objects, text-based, SWAT Editor, uses Deikto "linguistic toy interface" for input/output.
- DL: http://www.storytron.com/

The Storytron platform by Crawford & Mixon (Crawford & Mixon, Storytron Interactive Storytelling, 2008) is one of the most complete examples of Interactive Storytelling. It is character based and uses a verb based action system. "It is centred on artistic works called storyworlds. Each storyworld is a universe of possible narratives. The technology is comprised of four parts: SWAT, the Storyteller, Deikto, Sappho, and the Storyengine". SWAT is used to create new storywords, the Storyteller is the software used to play storyworlds, Deikto is the English-like language used by players to create the sentences that drive their actions, Sappho is the scripting language the author uses to create a storyworld and the Storyengine drives the computer characters AI. Storytron's main novelties are the use of verbs and objects in the construction of action sentences (see Figure 20). Storytron represents characters emotions with cartoon face emotion icons called "Emoticubes". These are driven by the characters mood taken from an array of data in their personality model.

Review:

The system has the potential to define many actions and stories, but is complex for the author, even with the SWAT editor to create new content; "many concepts are counterintuitive, because of some necessary backward thinking" (Axelrad & Szilas, 2010).

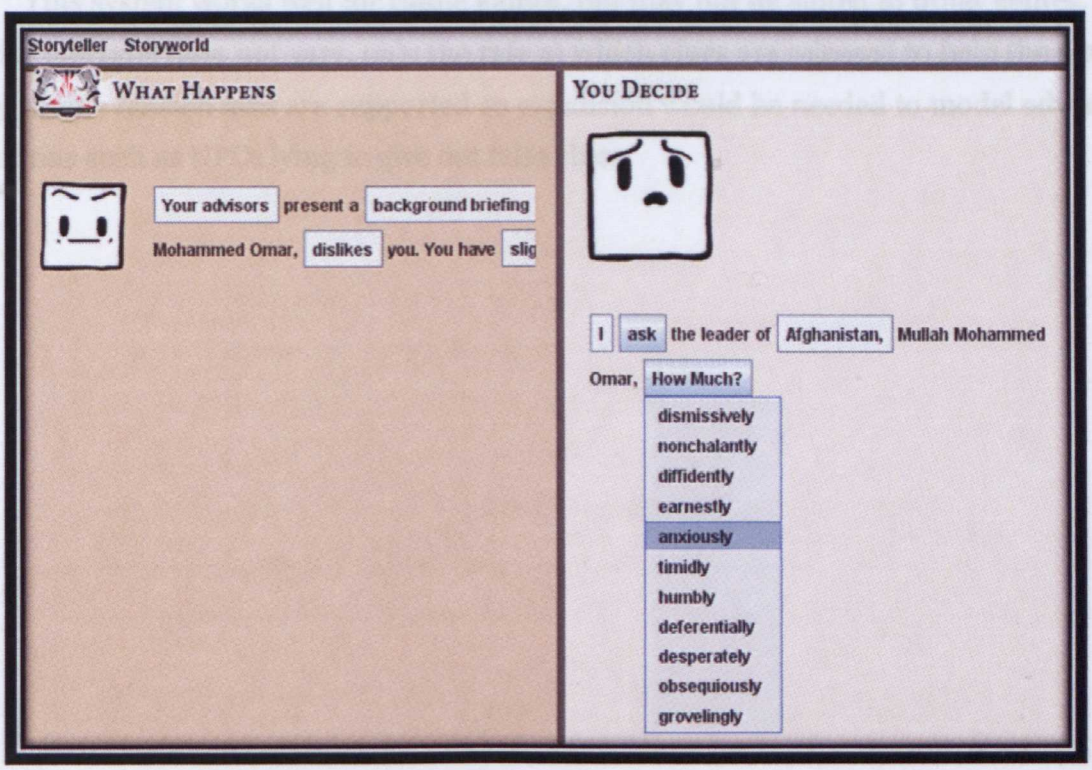


Figure 20: Storytron Screenshot

3.1.21 RIDDLE MASTER

Key Features:

- Planning based, tension arc, riddle based plots, Ugh's Story, Metric-FF, 3d world, and mouse-click action selection.
- DL: not available.

Barros & Musse create a system that focuses on 'Riddle Master plots' with pre-planned tension arcs (Barros & Musse, 2007). The tension arcs are used to model the player's current knowledge of the riddles events or the truth about what actually happened, with the story ending when the player solves the case. The tension arc can be modified to withhold or dispatch clues depending on the discrepancy between the author's desired arc and the player's actual progression arc; modelled as $D(t) = K*(t) - K(t)$, with K representing the increment, which is increased as each clue is released (with its magnitude equal to the clue's importance) and t equalling time. The clues are altered by generating new plans in Metric-FF for the NPC's, with a greater number of actions, causing them to work harder to give more new clues to the player.

When the generated stories closely follow an ideal tension arc, the narrative will progress in the pace imagined by its author (Barros & Musse, 2007).

Their prototype runs a 3D story world called Ugh's Story 2, with the player controlling the protagonist and making choices by clicking objects or characters with the mouse to produce a list of related actions.

Review:

This system works well for riddle games, but may not be suited to other genres. The stories outcome does not vary, only the rate at which clues are released to help the player. Only simple tension arcs are supported so expansion would be needed to model advanced situations such as NPCs lying to give out false clues.

3.2 NARRATIVE THEORY & MODELS

Narrative theory is the study of stories and how they are structured. By searching for a formula to describe the core components of a story and their grouping or order, whilst also keeping a level of narrative interest, the components can be served in a variety of combinations by changing the variables to create multiple unique stories. Interactive storytelling systems could incorporate some of these formal structures to allow computers to generate narratives and some publications covered in '3 RELATED WORK' have already experimented with various theories to great effect.

Some narrative theories are more formal than others, which only serve as a practical guideline but could still be useful. "Since our final goal is not theoretical but practical – 'How to improve [Interactive Storytelling] systems?' every approach, as soon as it is justified, is worth considering" (Axelrad & Szilas, 2010).

The following subsections explain some of these linear theories/models and include the author, compatible genres/story sets, related theories and systems already using the particular theory. These are divided into sections relative to the two main groups of thought:

- Dramaturgy (Drama Models)
  - **Aristotle's** Dramatic Arc
  - **Freytag's** Five-Act Model
  - **Field's** Three Act Paradigm
- Classical Narratology (Narrative Formalisms/Structuralisms)
  - **Propp's** Morphology of the Folk Tale
  - **Brémond's** Roles & Processes
  - **Souriau & Greimas'** Actancial Model
  - **Todorov's** Narrative Grammars
  - **Todorov's** Theory of Equilibrium
  - **Barthes'** Narrative Units
  - **Barthes'** Five Codes of Analysis
  - **Eco's** Theory of Possible Worlds
  - **Courtès et al.** Ethical Dimension
  - **Genette's** Discourse/Story Relation
  - **Campbell's** Hero's Journey

3.2.1 ARISTOTLE'S DEFINITION OF DRAMATIC ACTIONS

Aristotle's drama model was created around 350 BCE (Before Christian Era) in Greece and related to classic poetic narrative such as: comedy, tragedy, and epic themes. His rules can be transcribed into a high level algorithmic model, making them suitable for Interactive Storytelling.

In (Axelrad & Szilas, 2010) (Aristotle, 1961) tragedy is said to contain the following elements in order of importance: "Plot, Character, Diction, Thought, Spectacle and Song"; with their relationship defined in this fixed set of rules:

- "A plot is the list of incidents presented in a specific sequence."
- "An incident implies an agent."
- "An agent possesses character and thought."
- "An action implies character and thought."
- "From character and thought stems action."

The sequencing of the actions must follow these constraints:

- "The beginning action has no probable or causal action before it, but does have one following it."
- "The middle follows an action as a result of causality and gives way to another action as a result of causality."
- "The end is a causal result of an action and has no causal actions after it."

The overarching plot must also follow these general rules:

- "A good narrative must have a good plot, putting all the importance on action. The sequence of actions should arouse/inspire an emotion."
- "Plot with a change of fortune is Simple while Plot with a change of fortune through Reversal of Recognition is complex."
- "Reversal is an action's opposite. Recognition is a change in state from ignorance to knowledge."

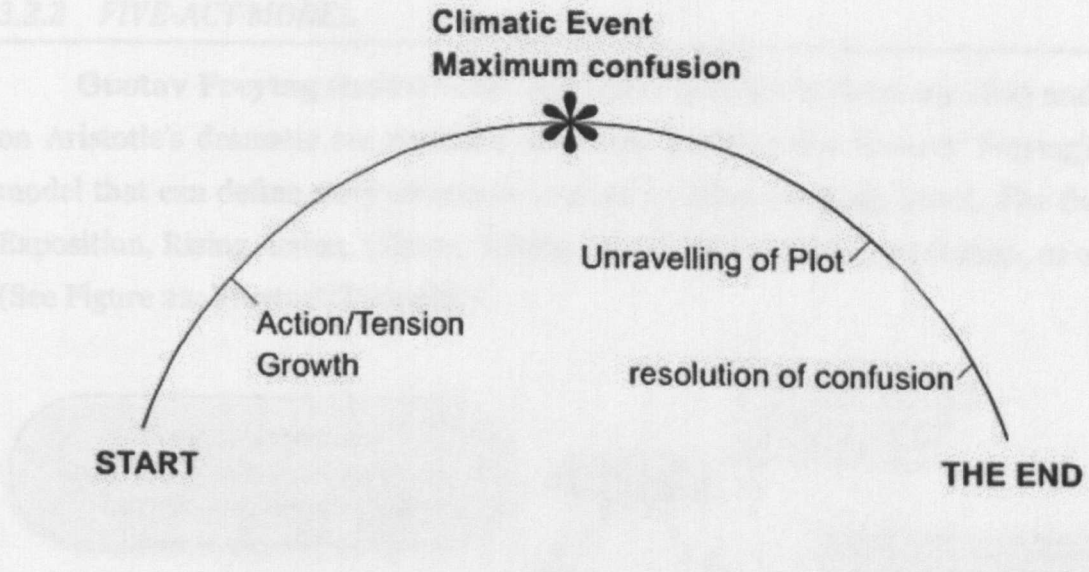


Figure 21: Aristotle's Dramatic Arc

Aristotle's rules can be used to create a dramatic story arc of tension, confusion, unravelling and resolution as depicted in Figure 21. These rules have been more recently revised by Tomaszewski and Binsted (Tomaszewski & Binsted, 2006); form the base of the 'Five-Act Model' and 'Three-Act Paradigm' screenwriting theories (covered in the next two sections); and have been incorporated into many storytelling systems such as DEFACTO (Chapter 3.1.6), FAÇADE (Chapter 3.1.10) and GADIN (Chapter 3.1.12).

Gustav Freytag studied Greek and Shakespearian in Germany 1863 and expanded on Aristotle's dramatic arc narrative structure, creating the five-act 'Freytag's Pyramid' model that can define story structure in plays or films (Freytag, 1900). The five acts are: Exposition, Rising Action, Climax, Falling Action, Dénouement, resolution, or catastrophe (See Figure 22: Freytag's Pyramid).

Figure 22: Freytag's Pyramid (Wheeler, 2004)

The weakness of this model is in its application to complicated modern plays that can contain multiple scenes (around 25 in some cases) that are difficult to pigeonhole into a clear five-act structure. Freytag's Five-Act Model heavily influences Syd Field's Three-Act Paradigm which is explained in the next section.

Syd Field describes his drama model, the Three-Act Paradigm in his book Screenplay (Field, 1979). He describes the core elements of a story as: actions, characters, scenes, sequences, Act I, Act II, Act III, incidents, episodes, events, music, and locations (Field, 1979).

THREE-ACT PARADIGM

Figure 23: Field's Three-Act Paradigm (Field, 1979) (Christopher, 2010)

Figure 23 shows that a screenplay should be made up from three main acts that setup the story (ACT I), add confrontation (ACT II) and bring resolution (ACT III). The illustrated plot points are incidents or events that allow the story to take a new direction, with the pinches designed to "pinch the story back on track" (Christopher, 2010) and focus the story on the next event.

This model focuses on the main characters quest that is a suitable format for computer games that traditionally allow the player to experience the story from a single characters point of view. Some Interactive Storytelling systems today can utilise aspects of the Three-Act Paradigm although they do not all enforce it, for example: DRAMACHINA (Chapter 3.1.7), which could use its Scene object to represent each act and IDTENSION (3.1.14), which has a similar "concept of obstacles" (Axelrad & Szilas, 2010).

3.2.4 PROPP'S MORPHOLOGY OF THE FOLK TALE

During the Russian formalism movement of literacy criticism and scientific study of poetic language in 1928, **Vladimir Propp** devised a system to describe the common components of 100 Russian folk tales. His morphology contains a set of 31 functions that have to appear in order, but don't have to be included in their entirety.

The functions in the correct order are: Absention, violation of interdiction, reconnaissance, delivery, trickery, complicity, villainy or lack, mediation, beginning counteraction, departure, first function of the donor, hero's reaction, receipt of a magical agent, guidance, struggle, branding, victory, liquidation, return, pursuit, rescue, unrecognized arrival, unfounded claims, difficult task, solution, recognition, exposure, transfiguration, punishment, and wedding. Propp also identifies 7 roles that the characters assume as the fable unfolds; the hero (who seeks something), the antagonist (villain), the donor (helps the hero by giving them an important object), the dispatcher (sends the hero on their quest), the helper (supports the hero), the false hero (pretends to be the hero), the princess or her father (needs protection from the villain and rewards the hero) (Axelrad & Szilas, 2010). Wee and Seifert (Wee & Seifert, 2001) created the web-app 'Proppian Fairy Tale Generator' that allows users to select which of the 31 functions are to be included, then prints a section of text selected randomly from several possible strings for each function that was used, generating the full story as a number of paragraphs. Propp's morphology has also been used previously in systems with a director agent, to create their user roles and also build story functions.

Propp's model can work in some cases with Interactive Storytelling but due to the pre-defined ordering is not suitable for a fully branched narrative. Other complaints include the model's "[failure] to recognise the importance of such story components as tone, mood, characterization, and writing style" (Wee & Seifert, 2001). As an example we can compare the narrative in films/movies (a more established format) with Propp's model. Hero's journey style stories like the Star Wars films fit perfectly, but more modern nonlinear films such as Pulp Fiction do not.

Later theories build on Propp's work and are covered in the next three sections (3.2.5 ROLES & PROCESSES, 3.2.6 ACTANCIAL MODEL and 3.2.9 NARRATIVE UNITS).

Claude Brémond a member of the French structuralism movement in 1965 built on Propp's work on analysing folk tales and dividing them into dramatic units (Bremond, 1966). Brémond grouped Propp's functions in a more flexible way and also requires the characters to have specific roles, defining functions for their actions and the effects on the overall story. The new updated model describes **Processes**, **Agents**, **Patients** and **Functions** in the following manner (Axelrad & Szilas, 2010):

- Actions are now called **Processes**.
- Characters can have the role of **Patient** or **Agent**.
- A **Patient** is affected by a **Process**.
- An **Agent** creates a **Process**.
- **Processes** are carried out by **Agents** or **Patients**.
- A **Function** is the relation between character and process and its overall effect on the story.
- A **Process** has three steps: **eventuality**, **action**, **result**.
- **Processes** can be ordered: **sequentially** (one after the other), **imbricated** (overlapping) or **parallel** (side by side).

Patients also have states associated with them, which can be changed by the **Processes**.

A = State of a patient (P) at time (t)

A' = State of (P) at time (t') ($Where\ t' > t$)

If $A == A'$ then either:

1. **No modifying** processes exist.
2. A **modifying** process exists but is **not complete**.
3. A **conserving** process exists and completed **successfully**.

If $A! = A'$ Then:

1. A **modifying** process exists and completed **successfully**.
2. A **conserving** process may exist but **failed**.

This allows characters to choose whether or not to deal with a situation and also models their resulting success or failure. The characters roles switch in a cycle between Patient and Agent.

Brémond's theory is one of several that drives the narrative logic in 3.1.14 IDTENSION.

3.2.6 ACTANCIAL MODEL

French structuralist **E. Souriau** and later refined by **A. J. Greimas** in the mid-twentieth century (Axelrad & Szilas, 2010). Following Propp's work they found a general model that could be applied to a wide range of stories. As a complete model it can describe simple narratives such as myths and quests. As an action model it can be applied more widely. The model is composed of the following six actants:

- The **subject**: the main actant and hero of the story.
- The **object**: that the subject is directed towards.
- The **helper**: helps the subject reach the object.
- The **opponent**: hinders the subject.
- The **sender**: initiates the quest between subject and object.
- The **receiver**: desires the object.

These are then split onto the following 3 axis:

- Axis of **desire**: subject/object, the main axis.
- Axis of **power**: helper/opponent.
- Axis of **transmission**: sender/receiver.

The distribution of actant roles can be split over multiple characters or one character can have multiple actant roles; actants can also shift between characters as the story progresses.

The Actancial model treats the narrative as a large sentence and would be useful for Interactive Storytelling due to its generativity. Switching characters and roles can create different takes on the narrative outline, but the model would need to be combined as a building block with other models to generate the narrative actions to go with the scenario, so to date few systems have used this.

3.2.7 NARRATIVE GRAMMARS

In the 1960's **Tzvetan Todorov** created a model also based on Propp's work, but with 3 narrative aspects (Todorov, 1969):

- **Semantic** aspect: story sense (content, values, messages).
- **Syntactic** aspect: The combination of narrative units and their relationships. The order of the sequence of narrative events is similar to Propp's work in 3.2.4 PROPP'S MORPHOLOGY OF THE FOLK TALE, but the sequences transitional rules are more general.
- **Verbal** aspect: the sentences used to tell the story.

"Todorov considers a sequence to be a series of linked narrative propositions" (Axelrad & Szilas, 2010). These narrative units are linked with a logical, temporal or spatial relationship; where one unit is the cause and another is the effect. These are described in (Todorov, 1969) as:

- **Proper name**: Agent.
- **Adjective**: Agents states.
- **Verb**: Agents action; defined further as:
  - **Mask or unmask**: while the situation might not actually be modified, someone is made to believe it is modified.
  - **Words** - calm a situation, hurt a situation.
  - **Physical** - attack or resistance.
  - **Seek help** - by evoking compassion or asking for advice.
  - **Change of location** – moving/walking.
  - **Exchange** - through payment or giving freely.

This model has great formal detail and would be fitting for the non-linear nature of Interactive Storytelling. It is partially used in the narrative logic of 3.1.14 IDTENSION.

3.2.8 TODOROV'S THEORY OF EQUILIBRIUM

Tzvetan Todorov was a Bulgarian literary theorist who suggested most narratives start with a state of equilibrium that is disturbed by an outside force, which must be battled against to return to an equal state.

Todorov's theory states that there are five stages a narrative can pass through:

- **Equilibrium** – a state of equilibrium.
- **Disruption** – a disruption by an event.
- **Recognition** – recognition of the disorders occurrence.
- **Repair** – an attempt to repair the damage.
- **New Equilibrium** – the return of a new equilibrium

These stages are not linear but can run multiple times in a circular cycle, with the new (different) equilibrium being reached changing the story each time.

Equilibrium -> Disequilibrium -> New Equilibrium

3.2.9 NARRATIVE UNITS

Roland Barthes the French semiologist suggested a more general model in 1968, which did not include structural rules, but had a broader concept of three principles:

- The story is built from narrative units, which all have a purpose.
- Each unit has a different 'type' that defines its function.
- The meaning of narrative units and their relations are organised hierarchically.

The types of narrative units are categorised into the following two main types, each with their own subtypes:

- **Functions** – these are the most important units and contain links to other functions, creating a chain of actions and events.
  - **Cardinal Functions:** open and close story possibilities.
  - **Catalysis:** expand cardinal functions to fill in the gaps.
- **Indices** – these expand functions providing further information.
  - **Pure Indices:** implicit descriptions.
  - **Informants:** explicit facts.

Cardinal functions are organised into unit sequences of related actions, with the highest level sequence being the main narrative. Changing these sequences will alter the overall narrative and changing the catalysis and indices changes the stories discourse.

3.2.10 FIVE CODES OF ANALYSIS

Roland Barthes the French semiologist suggested in 1977 that written narrative works with five codes which activate the reader to make sense of it. These codes are:

- **Action**: the characters behaviours.
- **Enigma**: how is the story knowledge disclosed to the reader?
- **Symbolic**: themes based on symbolism.
- **Semic**: the ideas/feelings suggested by the narrative.
- **Referential/Cultural**: where the reader is assumed to have common reference or cultural knowledge not explicitly stated in the narrative.

Some of these codes are used in many Interactive Storytelling systems, mainly actions and the concept of knowledge/enigma.

3.2.11 THEORY OF POSSIBLE WORLDS

The Theory of possible worlds was devised by **Umberto Eco** in the 1960's and was originally designed for written text. Eco states that narrative text is not complete without the reader and that the author must plan for all possible interpretations of their work. In Interactive Storytelling this could be used to generate a separate branched path for each possible interpretation of the narrative. The model of the user in 3.1.14 IDTENSION is based on Eco's 'Ideal Reader' concept.

3.2.12 ETHICAL DIMENSION

"The axiological or ethical dimension of narrative (how a narrative conveys ethical values) has been studied by various authors such as Courtès, Hamon, and more recently Adam and Jouve" (Axelrad & Szilas, 2010). Most narratives contain ethics, but this model is best suited to fables. The ethical values can be represented on bi-polar scales such as honest/dishonest.

The systems described in 3.1.6 DEFACTO and 3.1.14 IDTENSION use ethical dimension to compute the decision of dramatic actions using values.

Ethical dimension is also used in commercial games such as Fable and the Mass Effect series of games use a real-time conversation system with lifelike digital actors. The user chooses a shortened "emotional and instinctive response" text from a wheel menu interface rather than reading full lines and re-expressing them when they are said in the characters voice. The responses chosen by the player start to create branches in the conversation and therefore narrative. Some responses will have a moral value (either good or bad: paragon or renegade) and can cause both conflict and resolve in certain situations. There is also an interrupt action opportunity at various pre-determined conversation points with the paragon or renegade value attached to them, which can also be ignored by the player. The only problem with these representations is the black and white view of morals with a clear good and bad choice and no grey areas often found in real life.

Gerard Genette (Genette, 1983) analysed novels and fiction and broke narrative into three main levels: "Story" (list of events that happen), "Discourse" (how they are announced) and "Narration" (including the medium of conveyance). He also factored in the timing of the story and discourse levels and split this into three categories:

- **Order** – linear or flash back/forward.
- **Duration** – the ratio of story time to discourse time (how long the event lasts and how long it is presented for), which can make fast events appear slowly or can quickly breeze over long events that are not plot essential. For example a character's three years spent in jail may only take up five pages in a novel or 10 minutes of a movie.
- **Frequency** – certain events can occur multiple times (such as character habitual actions) or one event can be re-told again and again to reiterate its importance.

In Interactive Storytelling giving a player the freedom to choose their path makes it difficult to flash forwards and predict the future in a way linear stories can. Also many systems don't separate the story and discourse components. "The story/discourse theoretical distinction and analysis presented above, though rather simple in principle, has been so far largely overlooked in IS so far. This could open the way to interesting new approaches in IS" (Axelrad & Szilas, 2010).

Bae and Young (Bae & Young, 2008) have been working on a computational model for flashbacks and foreshadowing using Genette's theory, called Prevoyant.

Figure 24: Input and Output of Prevoyant (Bae & Young, 2008)

One of the oldest narrative patterns is found in ancient mythology around the world and was termed 'Monomyth' by **Joseph Campbell** in his book of comparative mythology 'The Hero with a Thousand Faces' (Campbell, 1949).

He stated that many myths share the same structure and 17 general stages, organised into three main groups; although some can be skipped, re-ordered or covered more intensely. "A hero ventures forth from the world of common day into a region of supernatural wonder: fabulous forces are there encountered and a decisive victory is won: the hero comes back from this mysterious adventure with the power to bestow boons on his fellow man" (Campbell, 1949). The Office of Resources for International and Area Studies list the breakdown of the Monomyth framework stating that "the stories and what they reflected of shared and diverse cultural roots of world history were exciting and fun to read and teach" (ORIAS, 2000). The Monomyth/Hero's Journey framework is apparent in parables, myths and fables, from King Author to Adam and Eve; but is still relevant to stories today and can be seen in popular film series such as Star Wars, Harry Potter, the Matrix and Rocky.

Figure 25: Outline of the Hero's Journey (Campbell, 1949)

3.3 COMPUTATIONAL MODELS

The main five computational models that have been used for Interactive Storytelling systems in the past are Bayesian Networks, ISRST, Linear Logic, Automated Planning and Scheduling. We will concentrate on the latter as it is the most advanced, developed and rich model and has been used in a diverse range of software solutions, from creating intelligent agents in computer games, such as F.E.A.R, to controlling robots, intelligent systems and even the Hubble Space Telescope at NASA. Planning is also the best fit for Interactive Storytelling as it allows the decoupling of actions and goals (Orkin, 2006).

3.3.1 AUTOMATED PLANNING AND SCHEDULING

Automated planning and scheduling is a branch of Artificial Intelligence that deals with the automatic generation of solutions to goals via action sequences, usually by intelligent agents or robots. In an automated planning and scheduling task the planner is given parameters coded in a formal language, including an initial world state and a final goal state that needs to be reached. It also has a list of actions that can be performed and which states they affect. The planner finds a solution by listing the actions to carry out, in order of execution, to get from the initial world state to the goal state. The actions contain a precondition: a world state specification which needs to be true before the action can be used and an effect: the state of the world after the action takes place.

Popular techniques used in modern planning systems are:

Forward chaining – a data-driven state space search, which looks ahead from the initial state in the inference rules until a true 'if' statement is found and then completing the statement's clause effect ('then'), until the goal is found. Forward chaining is suited to dynamic situations where the conditions will change.

Backward chaining – a goal-driven state space search similar to forward chaining, but working backwards from the goal state to find a matching 'then' clause and then following its opening 'if' statement and adding it to the goal list if the match is true.

These searches can be optimised for greater solution speed by using the hierarchical relationships between conditions such as in Graphplan (Blum, 2001), search heuristics such as Enforced Hill Climbing (EHC) and heuristics with translation to propositional satisfiability, such as in SatPlan (Kautz & Selman, 2006).

3.4 CRITICAL ANALYSIS OF INTERACTIVE STORYTELLING TECHNIQUES

In this section we will critically analyse the papers mentioned throughout the literature review by splitting them up into their component parts to see which techniques are best suited to provide the functionality we desire from our Interactive Storytelling engine.

From the key points made in our background research and as this is a computer games technology research project; we have decided to use a first person narrative running in real-time, similar to the approach taken by Façade (Mateas & Stern, 2003) and the narrative techniques discussed previously in '2.1 RESEARCH AREA'.

Creating interactive stories can be challenging as we are replacing human imagination with computer AI. In an article for ERCIM News Fairclough "The research field of Artificial Intelligence (AI) has encompassed a series of more or less discrete approaches including neural networks, genetic algorithms, expert (rule-based) systems, machine learning techniques such as reinforcement learning, and case-based (memory-based) reasoning. These are based on cognitive and biological theories and most are good at certain specialised types of tasks, much like different parts of the brain, and different people, are good at certain types of tasks. When it comes to creativity, however, all of these techniques are left in the dust by the human mind..." (Fairclough C., 2004)

To make our engine novel we will combine several of the techniques from other projects, such as PDDL planner/goal based characters (Barros & Musse, 2007), personality models (Crawford, Personality Modelling for Interactive Storytelling, 2004), a Story Manager (Fairclough & Cunningham, 2003), verb based actions and a point and click user interface with action filtering using preconditions in predicate logic (3.1.9 FABULATOR and 3.1.21 RIDDLE MASTER); whilst applying knowledge from other storytelling media, such as cinema, theatre and books (Noyle, 2006) (Murray, 2004) (Spector, 2007).

Sticking to one narrative theories strict model can narrow the author's narrative scope and story genre, whereas some models are far too general to use alone, so we will also combine the most useful aspects of the narrative theories covered in section 3.2. Table 1: The Importance of Narrative Theories in the DISE Framework, below, lists the narrative theories that have the most significance to an Interactive Storytelling system like the one we are proposing and references the most significant parts of each model that we will use.

The DISE storytelling engine will be object/character and verb centric, with respect to choosing the action and the object of the preposition. The story author's narrative

control will derive from structured scene elements that are triggered by predetermined rules, for example scene two can be triggered by the event: "dawn of day 2", to create a "narrative of actions" (Fairclough & Cunningham, 2003).

We intend to use key rule based, planning and heuristic methods to filter out actions that are irrelevant or inappropriate to the player at a given point. This method can be used to:

- Limit a player's choice to make them to stay true to the basic story world and its constraints.
- Avoid breaking the flow of the story by having commands & actions that don't work.
- Avoid making them think that they are restricted to set actions and a fixed path.

In Fairclough & Cunningham's system 3.1.15 INTERACTIVE STORY ENGINE, computer controlled non-player characters utilise the following five part model:

- Low level control behaviours – e.g. path-finding.
- Social simulation – gossip system.
- Idle behaviours – such as patrol area.
- Targeted behaviour – goal search and execution.
- Attitudes – characters score other characters that they meet or hear about and record events for later gossip.

This model is useful and would apply to many multi-agent systems our character system will also include low level controls, idle behaviours, targeted behaviour and attitudes via a personality model, but will be executing these systems using different models and heuristics.

Non-player characters actions are controlled & sequenced by the character engine, which will update their individual plans with new actions. Each character will have a personality model to record their current emotional state and views of other characters in the game. Each character has a list of actions that they can carry out that can be chosen from the same list that the player utilises, allowing the characters to be given loose roles based on their abilities (for example only the fireman character has the action extinguish house fire). By referencing specific personality traits, the rules embedded in each action can help the computer controlled characters to choose an appropriate behaviour.

When characters have no goals we will use Fairclough & Cunningham's idle behaviour concept from 3.1.15 INTERACTIVE STORY ENGINE "When an NPC does not have a goal to achieve, they can execute behaviour such as patrolling around a house or

following another NPC. These behaviours are assigned by the story author" (Fairclough & Cunningham, 2003).

Table 2 shows a comparison of storytelling techniques and why they're suitable or unsuitable for the criteria we wish to meet in the development of our own storytelling engine, editor and general framework.

Table 1: The Importance of Narrative Theories in the DISE Framework

Narrative Theory	Importance in the DISE Framework
3.2.1 Aristotle's Dramatic Actions	Agents have character and thought, from this stems action.
3.2.2 Five-act Model and **3.2.3 Three-act Paradigm**	Using scenes, story mods and story triggers in the DISE Story Manager (5.5 STORY MANAGER) the author has the power to create any number of acts and the transitions between them, allowing either of these models to be represented along with more modern non-linear models seen in films such as Pulp Fiction and Memento.
3.2.5 Roles & Processes	The DISE system represents agents, patients and modifying processes using characters and verb based actions along with planning data for their effects.
3.2.7 Narrative Grammars	DISE uses the syntactic idea of verbs and nouns to create the action sentences that later become events when carried out.
3.2.9 Narrative Units	DISE has a hierarchical decomposition of goals and actions and can allow the author to inject indices using scenes and the story mods fact based predicate logic.
3.2.10 Five Codes of Analysis	The 'Action' code is used in DISE, with the other codes cropping up as general narrative elements depending on what story is created.
3.2.12 Ethical Dimension	Author can use scenes and story triggers along with customised character personality models to recreate the ethical dimensions moral choice component.

Table 2: Interactive Storytelling System Review Table

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
ALTERNATE REALITY STORY GAMES 3.1.1	n/a	n/a	Presented using actors and modern media.	General public.	Can be large, international and spread across many media, e.g. internet, TV, radio and outdoor events.	Alternate Reality.	The users gather fragments of story information, solve cryptic clues, send messages and attend special events.	Most ARG methods have been explored using nearly every form of media.
ACTAFFACT 3.1.2	Resolution of conflict based on emotions, mentioned in multiple narrative theories.	BDI (belief-desire-intention) plan based reasoning agent framework. OCC and JAM BDI architecture.	2D graphical display using the Batik Toolkit.	Researcher	Created a research tool using an appraisal-based architecture for agents as actors.	Conflict based drama.	User's options to interact and alter behaviour are limited.	Requires direct modification of the programme base.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
BOVARY 3.1.3	Brémond's Roles & Processes.	STRIPS Heuristic Search-based Planning (HSP) System.	3D graphics using UT2003 Engine.	Adults.	The Bovary prototype seems useful for dramatizing scenes where characters discuss their feelings or make decisions based on emotions.	Madame Bovary Novel.	The user can interact with the characters in the scene by typing instructions in natural language text to direct their next actions or change an emotional state, but the user is not a direct character in the story.	The many dialogue options have to be created by the story programmer/autho r, was also extended with EmoVoice emotional speech input.
CAROSA: A TOOL FOR AUTHORING NPCS 3.1.4	n/a	Uses a Parameterized Action Representation system with HiDAC + MACES.	3D graphics using Ogre Game Engine.	Story Authors.	Simulates Crowds with Aleatoric, Reactive, Opportunistic, and Scheduled Actions.	Functional crowds.	Author NPC crowds using scheduling software to create goals.	CAROSA features some ideas that would transfer well into an Interactive Storytelling engine with further development.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
DEATHKITCH-EN 3.1.5	n/a	Depth-bound planner.	3D graphics using UT2003 Engine.	Computer gamers.	Allows the surrounding environment to act as a character in the story.	Urban Horror.	The player interacts by clicking objects in the scene and selecting an available action for their 3rd person avatar to carry out.	There is no authoring tool so programming new objects behaviours could be complex.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
DEFACTO 3.1.6	Aristotelian Arc Structure	First order logic.	Dated 3D graphics using VRML	Computer gamers.	Defacto's algorithm run in two main stages, the first one generating the sequence of actions for the characters depending on their goals and the choices made by the user; and the second one evaluating interesting situations and story outcome according to a set of dramatic rules.	Greek Tragedy.	The user moves about in a 3D-VRML scene and at key moments chooses from a set of multiple choice text options.	Complicated algorithms and lack of authoring tools make it very hard for researchers to expand on.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
DRAMACHINA 3.1.7	Bremond's Roles and Processes, Three-Act Paradigm and Ethical Dimension system.	File directory model.	Text and GUI menu based editor.	Writers.	DraMachina is a valuable tool for writers to author narrative concepts in structured XML, which can be interpreted by programmer.	Narrative authoring tool/any.	Writing classic narrative and assigning character data.	Only in prototype form and would take time to extend and develop further; although some of the structural concepts (such as the definition of scenes, areas, actors and objects) could be borrowed.
DUNGEONS & DRAGONS AND THE GAMES MASTER 3.1.8	Dungeon Master controlled story.	n/a	Board game.	Gamers and families.	One of the best examples of true interactive storytelling, difficult to replicate with computer AI.	Fantasy RPG.	Either dungeon master or hero character.	Extendable using custom user generated quests and official add-on packs.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
FABULATOR 3.1.9	Classical 'three act' narrative structure.	Planning algorithms and STRIPS domains with A* search.	3D.	General users.	Ugh's Story, is the base system for 3.1.21 Riddle Master.	Mystery/ Whodunits.	The player executes actions with the mouse: clicking in an object (or character) brings a list of actions related to it.	FABULATOR has source files for the planner which can be modified but no editors or source code distribution.
FAÇADE 3.1.10	Aristotelian Arc style partially ordered drama.	ABL (A Behaviour Language), NLU (Natural Language Understanding), Jess RBS.	3D using OpenGL.	Adults.	Façade is the only published and most complete interactive storytelling system.	Relationships.	User has to interact with two NPC's that are having marriage problems, using the mouse & keyboard and inputting natural language text.	Commercial - Not designed to be modified so is a closed system with no authoring tools.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
FEARNOT 3.1.11	Forum Theatre.	Uses the OCC Model for character/agent emotions.	3D characters with chat window overlay.	School children and teachers.	The FearNot system is novel in its ability to create unscripted autonomous characters using double appraisal and its use in schools as a bullying simulator/tutorial.	Bullying education.	The user watches a clip of a bullying scene and then talks to the victim using a text-chat window to advise them on what to do next.	Players are invisible friends that can only interact with characters at certain points via messenger style text chat-bot, so the source code would need heavy modification to add the player as a character.
GADIN 3.1.12	Dilemma based narrative based on Aristotle's dramatic, 3-Act Paradigm and 5-Act Model.	Planning and Sequencing.	2D GUI menu/text based.	Computer users.	Narrative dilemmas allow on-going story generation with no definitive end.	Soap Opera.	User input a choice to resolve each dilemma.	GADIN has no tools to create new stories so would need to be hard coded.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
HEFTI 3.1.13	n/a	Evolutionary/ Genetic Fuzzy Algorithms.	2D using Microsoft Agent and windows menu UI.	Story Authors.	HEFTI generates dynamic stories from a set of authored story constructs given by human authors; a set of authoring tools that allow authors to generate the needed story constructs; and, a storytelling environment for them to orchestrate a digital stage play with computer agents and scripts.	Fairy tales/ Plays.	Users input narrative fragments and HEFTI will select, arrange and mutate them to dynamically create the narrative.	HEFTI has an integrated Authoring Environment (IAE) and a Graphical Object Library to create new stories with.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
IDTENSION 3.1.14	Narrative Grammars, Roles and Processes, Three-act paradigm, Five-act model.	First-order Logic/rule based systems.	3D scene with text input.	Computer users & computer gamers.	IDtension is an Interactive Drama engine where the narrative unfolds as the user decides what actions the main character will perform in relation to the other characters in the story.	Historic sea-faring/mutiny.	The user decides the actions that they would like the main character to perform using a text menu interface.	The authoring of stories is difficult.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
INTERACTIVE STORY ENGINE 3.1.15	Propp's morphology of the folktale.	Case-based planning and story Director.	3D graphics	General users.	Uses an expert case-based character director system which dynamically generates and controls a story, which is played out in a multiplayer networked game world with parallel stories.	Folktales.	Users explore and click on interactive objects in the 3d world, choosing action to change the story.	To create new stories the author must write sequenced move descriptions using a scripting language based on Propp's roles and processes.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
LOGTELL 3.1.16	Propp's morphology of the folktale.	Goal-inference planning.	3D.	Story dependant.	The system uses goal-inference rules, planning and user intervention to drive a 3D drama visualisation that can be used for entertainment or story creation within a specified genre.	Folktales / Fables.	The user watches the story from a third person perspective, with indirect/passive interaction.	LOGTELL has no tools or editing available.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
S-MADE 3.1.17	n/a	Goal Net, Fuzzy Cognitive Goal Maps, Drama Manager.	Active Worlds 3-D virtual environment powered by Renderware.	Computer users.	S-MADE uses Fuzzy Cognitive Goal Net to create, dynamic storylines as well as character behaviours simultaneousl-y in real-time.	Mystery.	In the Storytelling process, the audiences are able to affect the decision of director as well as single character.	A scriptwriter creates the story plot which contains multiple storylines. Then the story plot is sent to a director agent for execution.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
SIFTABLES 3.1.18	n/a.	n/a.	2D pictures.	Children and parents/ teachers.	For the storytelling game several Siftables each depict an object or character which when moved around or connected together change the state of the story on the main view screen, which displays the whole scene.	Children's stories.	The user can move the Siftables around, connecting together characters and objects to generate a story.	The development kit is available for the Siftable hardware, so other storytelling applications could be produced.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
SLEEP IS DEATH 3.1.19	Similarities to a human Dungeon master?	n/a	2D pixel based.	General users, some computer knowledge required for editing.	Sleep Is Death is a 2D storytelling game for two players that blur the line between playing and creation of stories.	Any.	One person fulfils the role of the 'player' with the other acting as the 'controller'.	Editing tools allow the narrative to be advanced in two minute turns. User generated asset libraries online and sprite editors extend the games content beyond any other IS system.
STORYTRON 3.1.20	Verb based.	Planning.	2D Menu GUI based with emotion icons.	General users.	The Storytron platform is one of the most complete examples of Interactive Storytelling. It is character based and uses a verb based action system.	Demo is balance of power – world leader simulation.	The user chooses options from a dropdown menu in response to story events.	The Story World Authoring Tool "SWAT" can be used to create new storyworlds, but is complex.

Literature/ Chapter	Narrative Theory	Computational Models	Graphics	Audience	Scope	Genre	User Roles	Extendibility
RIDDLE MASTER 3.1.21	Tension arc.	Planning using Metric-FF.	3D world.	General users.	The tension arcs are used to model the player's current knowledge of the riddles events or the truth about what actually happened, with the story ending when the player solves the case.	Riddles, mystery.	The user clicks objects and characters and chooses an action from the provided list.	The planning files are text based so can be modified, but there are no authoring tools and parts are hard coded.

3.5 CHAPTER SUMMARY

In this chapter we first looked at the approaches used by other researchers to create an interactive story and summarised their work in Table 2: Interactive Storytelling System Review Table (pages 84-97). From this work we observed the potential for further research into an Interactive Storytelling framework which has a greater focus on allowing the user to play as a central character in the story, rather than an observer. We also noticed that many of the systems do not have an interactive 3D environment and also lack the editing tools to create these environments as well as the narrative elements of the story world. It is also important that these editors are usable by writers and not just programmers who can modify source code.

Next we compiled a list of relevant narrative theories and models that could prove useful in creating a structure for our story to follow and a way of generating an interesting story. From these we picked out the key parts as described in Table 1: The Importance of Narrative Theories in the DISE Framework (page 83). Some of these narrative theories are key parts of our storytelling framework, such as verb based actions and characters with unique goals and personalities. Other narrative structures are optional and up to the user creating the story to follow, such as the act/scene structure (3-act, 5-act or more).

Finally we looked at computational models for Interactive Storytelling, mainly automated planning and scheduling with various search algorithms and goal driven actions.

The next chapter, 'DISE: AN INTERACTIVE STORYTELLING FRAMEWORK' introduces our take on interactive storytelling and documents the design of a framework "DISE: the Digital Interactive Storytelling Engine", that matches our requirements and specifications, such as: data types, points of view, game technologies and user characteristics. It gives an overview of the various components of DISE and includes their UML and class diagrams.

> *[Interactive Storytelling is] any game featuring both characters and a story in which one or more narrative aspects changes interactively (IGDA, 2001).*

4 DISE: AN INTERACTIVE STORYTELLING FRAMEWORK

4.1 INTRODUCTION

In the previous chapters we have looked at why storytelling is important, explored the relatively new field of Interactive Storytelling, and highlighted relevant work already carried out in this area. The main focus of this section is to determine how Interactive Storytelling (IS) systems can be improved and look at what makes a story interesting and immersive. We will break down the problem, the deliverables and create a framework of our approach in picking the important ideas to build DISE. It also contains our plan of development to show we approached the problem systematically, whilst raising any design issues that could occur.

In the previous section 3 RELATED WORK we identified the key elements from previously implemented Interactive Storytelling Systems and found that although some elements have been implemented, no one system has all of the following:

- The **player is a main character** in the story and can move and interact freely with the world to make decisions, which have an effect on the narrative.
- Full **3D graphics and animations**.
- Flexibility to create stories in **multiple genres** which are also not bound to one strict narrative theory for their discourse.
- **Intelligent non-player characters** which have their own personal goals and personalities.
- A **Story Manager** to constrain and progress the narrative in the ways that the author defined depending on certain events.
- Easy to use **editing tools** to allow new story content to be created.
- **Procedurally generated environments** which can be controlled with high level constraints.
- Character **facial animation** system with an editor.

- The ability to run **cross platform** and via **web deployment**.
- **Modular framework**, which can be expanded, upgraded and improved, for example: by switching the planner out for a newer version or a different planning system/algorithm/heuristic altogether.

Our Framework is called **DISE – Digital Interactive Storytelling Engine** and includes all of the features listed above. It is designed not around one strict narrative theory, but to allow the story author to control how they compile their narrative structure using the Scene Editor.

The **Story Manager** will monitor this narrative structure composed of multiple scenes and progress the narrative by making the changes that the author implements using the powerful **StoryMods** which trigger pre-defined rules written in predicate logic to update the state of the world. Using **Story Triggers** the author can choose when, how and why the current scene changes and which scene is next. A Story Trigger connects together two scenes, creating a one-directional from-to relational link and contain pre-defined rules written in Boolean logic that if returns true will move the current scene forward to the next one.

The **Character Engine**, which runs in a separate thread, sequences characters behaviour in turns with more important characters prioritised higher in the list. Each computer controlled non-player character (NPC) gets one turn per update loop cycle, which allows them to run an instance of a planner to solve a goals requirements and to carry out an action or continue and progress their current action. The planner will queue up a sequence of actions for the NPCs to execute on a turn by turn basis to complete their goals. The Character Engine also has systems for re-planning and prioritising goals and setting NPCs to wandering mode when they are idle and have no goals or actions, so un-programmed extras will still appear alive and busy, until they receive new orders.

The **Player Action Engine** handles all the player input systems such as mouse picking and keyboard movement. It is also responsible for allowing the player to interact with the world by choosing actions. When an interactive object is clicked the Player Action Engine uses context filtering to display all the relevant action verbs in a floating text cloud display around the object. It also executes the chosen action and updates the world with its effects.

In the next section we will look at the current games technologies needed to create and render these 3d worlds and handle the players input by updating what they can see on the screen.

4.2 STORYTELLING GAME TECHNOLOGIES

Storytelling via digital entertainment is at the forefront of many industries. It is entwined within the filming industry to create special effects. For example, it was extensively used to create the Beowulf production. Despite modelling the actors in digital form as they are in the real, it was techniques borrowed from the gaming industry that allowed digital representations of themselves to do things they couldn't do physically. As time goes on we see mass convergence between social networking and virtual environments which provide technologies that exceed conventional websites.

For example, many gaming or virtual environments provide a means of expressing oneself through the way you look and the artefacts you surround yourself with, as well as providing far more advanced social settings. Given, what we believe is a new and emerging phenomenon; it is not too difficult to imagine the Internet itself porting many 3D functions to represent content in a more life-like fashion. The challenge is to provide tools that simplify the production of 3D gaming and virtual environments to allow anyone to contribute to the production of the environment and the story content within such environments.

We are already witnessing a trend emerging, so to a certain extent this is happening. Moving from conventional modding, through content created by users within environments that support such capabilities, we see platforms, such as Xbox Live, that now provide a platform, where recreational game developers can create and deploy their work. Nonetheless, this is not performed from one single development environment, or for cross-platform developments. We addressed this problem through an integrated game development environment we have developed called Homura. The Homura IDE is a platform designed to facilitate the development of 3D Java games in an all-inclusive interface, which provides text and graphical editors, compilers and launchers, and interfaces to objects in the game (Carter, Cooper, Dennett, & Sabri, 2008) (Dennett, et al., 2008) (El Rhalibi, et al., Homura: A Step Further Toward 3D Java Game Development Support, 2008).

We have utilised the advances made within many existing technologies to create a new and novel platform where all aspects of gaming can be realised. This provides obvious benefits. First, it provides a single environment, which contains different development views dependent on what part of the game you are developing, thus the user does not have to learn and use many different applications. Secondly, it provides a set of simplified tools that can not only be used by professional game developers but also by less experienced developers. In this way we can tap into the imaginations of anyone connected to the Internet.

To create procedural stories we will use this game technology as the medium of conveyance as computer games are the most advanced form of interactive entertainment. Games technology has quickly evolved over the last 30 years and can now define a 3d scene in a fairly realistic or artistically stylised manner, with object creation and management and support for multiple input devices. In the following sections we will describe some of the desirable features needed to display a story world in real-time 3d and allow the player to interact with and navigate the world.

4.2.1 RENDERER

The renderer draws everything in the scene that the camera can see by generating an image from the scenes objects. In 3d graphics the renderer has to generate a 2d image from a scene represented in 3d by sending calculation data to a graphics card to solve the rendering equation. These calculations can vary but mainly include predicting the behaviour of virtual light and the effect it has on objects with various material properties. On top of drawing the basic scene geometry, the renderer processes the art pipeline to produce the following advanced functions and effects:

- Shading – adjusting the objects surface colour to be lighter/darker depending on the light source.
- Texturing – the mapping of 2d images to 3d objects to give greater detail and colour information.
- Bump mapping – adding fine detail by simulating bumps and wrinkles on an object by perturbing its surface normals then using these new values to calculate its illumination.
- Shadows – calculating where light is blocked.
- Transparency – making objects that are see-through.

- Reflections - for light bouncing off shiny surfaces like mirrors or water.
- Refractions – bending of light through clear objects like glass or water.
- Caustics – like indirect illumination but with the light highlights caused by reflection & refraction.
- Volumetric fogging – for a thicker environment like smoke.
- Indirect Illumination – global illumination, with light bouncing off other surfaces, not just emitting from the main light source.
- Subsurface Scattering – scattered transmission of light through semi-translucent objects like human skin.
- Bloom & HDR – an imaging artefact of real world cameras and to some extent the human eye, where very bright light sources in the scene bleed beyond their natural

borders. It can also be used for the transition from a dark to light room to represent human eyes adjusting to the new light levels.

- God Rays – columns of light radiating from a single point in the sky through gaps in clouds, tree canopies or broken cover.
- Motion Blur – fast moving objects or camera movement creates a blur.
- Depth of Field – things further away lose focus and become blurry.
- Anti-aliasing – smooth's the jagged edges of angled or curved 3d objects.
- Non-photorealistic Rendering (NPR) – artistic techniques to give objects a particular stylised look/theme such as cartoon shading, comic book halftone & outlines, or high contrast black and white.

4.2.2 SCENE GRAPH

As aforementioned, the main task of a 3D engine is to maintain and display 3D objects. A collection of 3D objects in a game is known as a scene. To relate all these objects, a so-called scene graph is used.

The scene graph is a graphical data structure that logically manages the spatial representation of all the scenes objects. Everything is arranged in a graph/tree of nodes, where nodes have many children but only one parent. Any operations or settings on a parent get propagated down to the children. This is useful for adding render settings, materials, textures, lighting, or spatial transformations to batches of grouped objects. These transformations could be used to move a race driver along when the car node moves or rotate planets around a sun. Using HUD node a collection of quads can be nested to create a user interface, which scales with the display resolution.

The scene graph allows a large level scene graph to be created, without having to process it all at once. Only the regions that are being observed are of direct interest, while scenes that are further away have a reduced amount of observable detail in areas, such as meshes and textures. This general concept is commonly known as spatial subdivision, with the graphical aspect being known as level-of-detail (LOD) management. The trade-off for this reduction in processing is more memory usage, and more code required for these aspects of the engine.

4.2.3 INPUT SYSTEM

The input system allows devices such as mouse and keyboards or game controllers to provide input data to trigger game logic. Listener objects can poll devices to see which ones are attached and being currently used. These listeners also provide trigger call-backs when an event happens. The event can be used to call any function and take the input values into account, such as mouse position, key pressed, key released and

left/right/middle button mouse clicks.

4.2.4 ART PIPELINE

It is important to have a useable art pipeline (Figure 26) to add new assets directly into the game world. Tools such as Adobe Photoshop and Autodesk 3DS Max can be used to create game data files for images/textures and 3d models respectively. The images and models have to be in a format that is readable by the asset manager classes and without strong art tools along with exporters and loaders creating something more complicated than a colour fill or simple geometric 3d shape can be tricky.

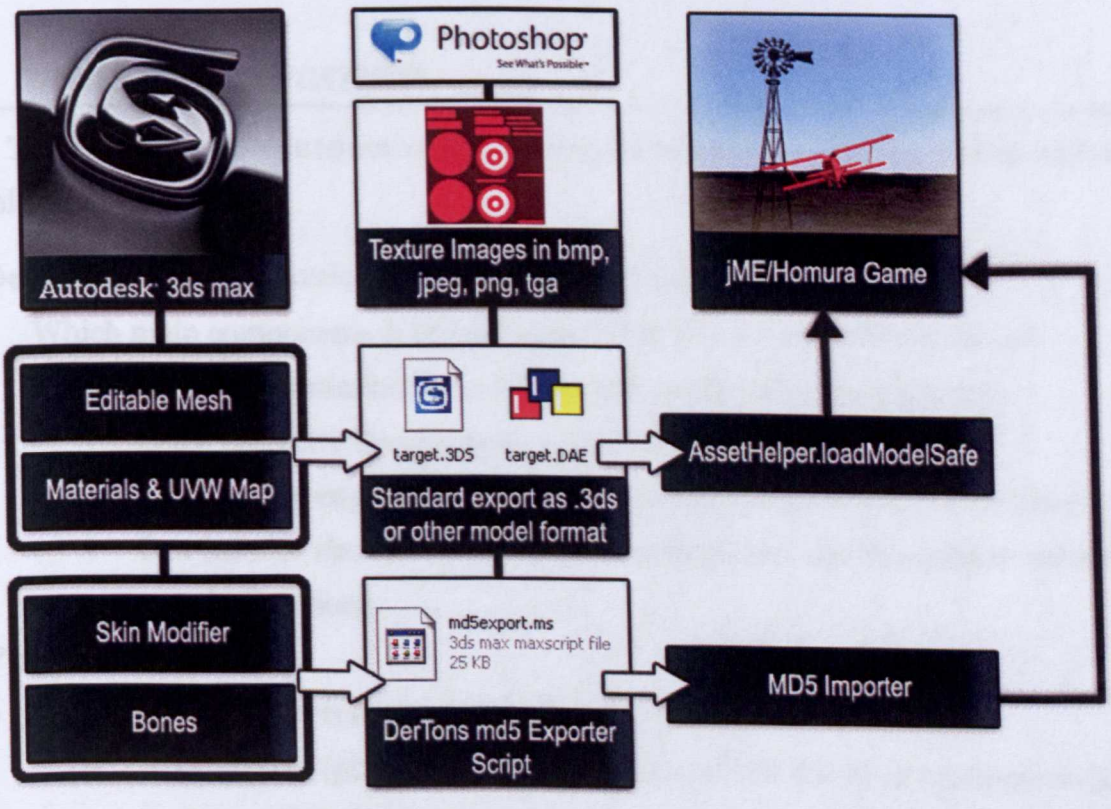


Figure 26: Homura Art Pipeline

Now that we have determined the requirements for the game engine used in DISE we need to identify the requirements of an interesting story and what systems will be necessary to convey this story, whilst adapting it to the player's choices and how these can be combined together to form a complete Interactive Storytelling framework.

4.3 REQUIREMENTS & SPECIFICATIONS

The first thing to think about is the basic needs of a story; usually there is a main character/protagonist that has a problem to overcome or a need to solve (see 3.2.14 Hero's Journey above). Looking also at Propp's 7 character roles gives a suggestion of this plus the other supporting characters usually found: "the hero (who seeks something), the

antagonist (villain), the donor (helps the hero by giving them an important object), the dispatcher (sends the hero on their quest), the helper (supports the hero), the false hero (pretends to be the hero), the princess or her father (needs protection from the villain and rewards the hero)" (Axelrad & Szilas, 2010). It is clear that a system is needed to easily manage and sequence seven (or more) characters with individual roles (goals) and personality traits.

The core physical elements that work together to create a narrative can be broken down into the following in game objects/classes: An audience, memorable characters, a location/setting for the action to take place in, actions for the characters to perform and objects of importance to interact with (props).

4.3.1 GENERAL SPECIFICATIONS

To create a capable Interactive Storytelling framework we need to find an answer to the following questions:

Design Strategies, Architecture & Game Engine

- Which main components & technologies is the framework made up from?
  - Which computational model can process the story interaction?
  - Can a narrative theory create a model to represent the story?
  - Which game engine can render and update the 3d world in real-time?
  - Can infinite choices be put into sentences and can the author control the story's direction?

Point of View

- Which point of view is the story told from?
  - A subjective (player has a limited view of the story) or omniscient (player sees everything) point of view?
  - Will the player control a single or multiple character(s)?
  - Will the player view a scene from a different perspective at certain points?
  - What perspective will the camera show the story world from (1st person/3rd person/whole scene)?

Actions

- How are actions defined?
  - When can an action be performed?
  - What does it interact with?
  - What happens after the action is carried out?

Characters

- How can we make believable characters which progress the narrative?
  - What do they look like?

- How do they behave?
- What are their intentions and desires?

Locations

- How can different story locations be created?
  - How can authors draw and import 3d graphical assets?
- What objects can be interacted with?
  - What do they look like?
  - What can a character do with them?

World Taxonomy

- How can the story world and everything in it be categorised?
  - How do we differentiate between a character and an inanimate object?
  - How do we differentiate between an object that is interactive and one that is just visual decoration/scenery?

Story Manager

- How do the story authors constrain the narrative to retain their original ideas?
  - How is the story broken down and revealed (three-acts, five-acts, and tension/suspense)?
  - How can the author introduce the hero's problem to overcome or create a conflict?
  - How does game model compare to narrative model?
  - How can new stories be created?
  - What editing tools are needed?
  - How is the story data stored on the computer?

4.3.2 IS FRAMEWORK DESIGN STRATEGIES

There are different strategies that need to be considered in order to tackle the problem of creating an Interactive Storytelling framework, some have been tried before and do not work, whilst others have much more scope for future work and expansion. The main differences between these are the richness of stories that can be produced, the effect of the players actions on the outcome of the story, the amount of content that has to be created and predefined and the amount of content that is experienced by the player.

Crawford lists some of the main strategies as: Simple, Environmental, Data Driven, and Language Based (Crawford, Chris Crawford on Interactive Storytelling, 2004). For convenience we will group these along with all of their sub-categories into the following two categories: **Incapable Strategies** and **Potential Strategies** based on their proficiency and stating why they lack capabilities or have the potential to work.

These are the storytelling strategies that have been tried in the past, don't work for true Interactive Storytelling and have no suitability or room for improvement. These are:

Branching Trees: As we have mentioned before in the background section; branched narrative usually splits each decision into a binary tree, leading to a new choice and further child branches (Figure 27). The problem with this technique is that content for the branches quickly spirals out of control with an extra 2^n pre-determined unique nodes on the tree for each of the 'n' choices a player is allowed to make and the player only experiencing one path through the story. For example the small story in Figure 27 with only 3 choices will lead to $1 + 2^1 + 2^2 + 2^3 = 1 + 2 + 4 + 8 = 15$ created scenes in total, with only 4 scenes (around 27% of the game) being experienced on each unique playthrough. The number of branches could be increased to create more choices at each node and making a richer story, but this would result in even more states/scenes to program, so is not a scalable solution.

Foldback Schemes/Converging Trees: These are similar to the branching trees above, but to limit the number of nodes in the tree some branches converge to the same point. This means that the player is given a choice to make which ultimately doesn't affect the outcome of the story, for example in Figure 27 the converging tree shows that both decisions leading to 'A' & 'B' eventually merge back to scene 'C', giving little divergence to the overall story plot.

Kill on Stray: Similar to the branched tree narrative but terminating branch numbers by having multiple choices with only one 'correct' choice and the others resulting in death of the player character, forcing a restart and for the player to learn the one correct path through the story. In Figure 27 choices 'B', 'C' & 'E' all lead to game over states, leaving only one true path to navigate along a single fixed story.

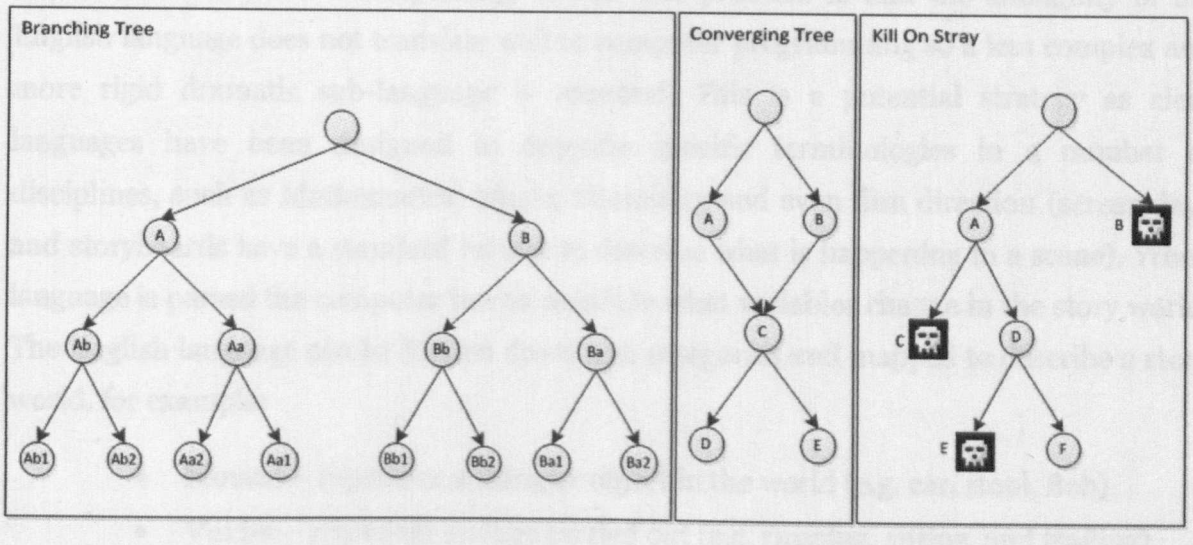


Figure 27: Branched, Converging and Kill on Stray Tree Based Narratives

Interleaved Story/Game: These are games that have elements of a story interleaved between gameplay sections. This can be presented in multiple ways including: written background story in the game's manual; story cut-scenes triggered after successfully completing a puzzle, section of the game, or level; pieces of story hidden in the game world that reveal more of the plot as they are uncovered; and 'in-engine' cut-scenes using scripting which solves the disjunction from cut away movies, but still are not changed depending on gameplay other than win/lose situations.

Environmental Strategies: These are games with a large 3D world usually inhabited by AI agents or non-player characters (NPCs). The player can further the narrative by locating and talking to these NPCs who provide personal anecdotes, nuggets of information, useful quest items or offer their assistance. Most games in the last few generations of PC and console hardware use this model and have been successful games, but are not true interactive stories. This model could be expanded on by creating smarter agents using Personality Modelling (see Section 5.8.4) or adding a narrative structure such as the Hero's Journey (see Section 3.2.14), but then these could be used independently of the Environmental Strategy, so are considered individually as entirely different strategies in the next section.

4.3.2.2 Potential Strategies

These are the strategies that have the potential and richness to create a true Interactive Storytelling system. These are:

Language Based Strategies: are inside-out approaches, which use words or symbols/images and grammar as their core model to represent drama. Crawford states his belief that "language itself contains the core elements of drama" (Crawford, Chris Crawford on Interactive Storytelling, 2004). The problem is that the ambiguity of the English language does not translate well to computer programming so a less complex and more rigid dramatic sub-language is required. This is a potential strategy as clear languages have been designed to describe specific terminologies in a number of disciplines, such as Mathematics, Music, Chemistry and even film direction (screenplays and storyboards have a standard format to describe what is happening in a scene). When language is parsed the computer has to calculate what variables change in the story world. The English language can be broken down into categories and mapped to describe a story world, for example:

- **Nouns** – represent a thing or object in the world (e.g. car, stool, Bob).
- **Verbs** – represent actions carried out (e.g. running, sitting, and trading).
- **Prepositions** – function words to describe temporal, spatial or logical

relationships (e.g. on, of, in, at, has).

- **Pronouns** – can replace nouns to select multiple types of object (e.g. we, they, and everyone).
- **Adjectives** – add to noun or pronoun to describe, identify and quantify (e.g. instead of car we could use 'small' car).
- **Adverbs** – can modify verbs, adjectives, whole phrases or can be stacked on top of another adverb to define the manner, time, place, cause, degree and add context of the scene (e.g. quickly, boldly, gently).

The language's grammar can be defined in many ways, the simplest one being a word matrix that lists every word along both the x and y axis, with the grid cells being marked true if the words can be connected as: '$y + x$'.

e.g. (y) Ride + (x) Bike = true, (y) Bike + (x) Ride = false, (y) Ride + (x) Tree = false.

A more complicated extension of this concept of cross-linking words is a **semantic network**, the biggest being 'WordNet' (Princeton University, 2011). WordNet is "a large lexical database of English, developed under the direction of George A. Miller (Emeritus). Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations" (Princeton University, 2011). Using the extra linkage dimensions a richer language can be produced and user input can be improved using **inverse parsing**. This technique allows one word from the user input to be read by the computer at a time and displays all the new possible choices for the next slot in the sentence. This narrows down choices, eliminates errors or dead ends and speeds up processing the whole input sentence, as it is done in small chunks whilst calculating the next possible steps. Inverse parsing has been used before to quickly display possible results in Internet searching and e-commerce product searching to save time so is a tried and tested method.

Data Driven Strategies: need a set of story data and model to describe how that data is combined into an Interactive Story. Crawford calls these "Story Components" and "Connectivity Data" (Crawford, Chris Crawford on Interactive Storytelling, 2004).

- **Story Components** are broken down pieces of story data that can be used as building blocks for an overall narrative.
- **Connectivity Data** describes how these are connected together and which pieces are used in response to a player's choices and actions.

To create the Story Components and Connectivity Data we can look at the work done in the field of narrative theory over many years by many famous scholars. Narrative theory is the study of "the devices, strategies and connections governing the organisation of a

story (fictional or factual) into a sequence" (O'Sullivan, Hartley, Saunders, Montgomery, & Fiske, 1994). Using these cross genre conventions and formalisations we can form the basic components of our story and the organisation of these into a narrative. To represent this data in a simulation the narrative theory will have to be paired with a computational model. The model will specify how the story components are stored in the computer's memory, their structure and the methods to monitor the story and adjust it by choosing the appropriate data information at the correct time. To find this data quickly (in real-time) whilst the story game is running a search heuristic/algorithm is also usually required.

Using different formalisations can dramatically change the outcomes, design and purpose of an Interactive Storytelling system, so it is important to evaluate different approaches in relation to our original project specification and the goals we want to achieve. Combining narrative theory with a computational model allows the aforementioned data driven and language based strategies to function and has the potential to create a compelling story that dynamically changes depending on a player's actions without the tedious work of hardcoding the specific connections between every outcome, including the ones that are never used.

4.3.3 STORY DATA ELEMENTS

To design a storytelling framework and create data flow diagrams the data dictionary needs to be identified. This is a list of the data elements needed in the software and how these can be combined into data records and stored on the computer.

- Physical Elements (who? what? where?)
  - Characters
  - Props
  - Locations
- Narrative Elements (what happens? when? why?)
  - Actions
  - Goals
  - Initial State of the Story
  - Predicates (Boolean states representing the current condition of the physical story objects.)
  - Scenes (Can force a particular event(s) to happen, progressing the story as the author desires.)

Figure 28 shows the data elements and types stored in the DISE database (which will be inside the story manager section of the framework). This data can also be split out to create a full PDDL file for the characters planning.

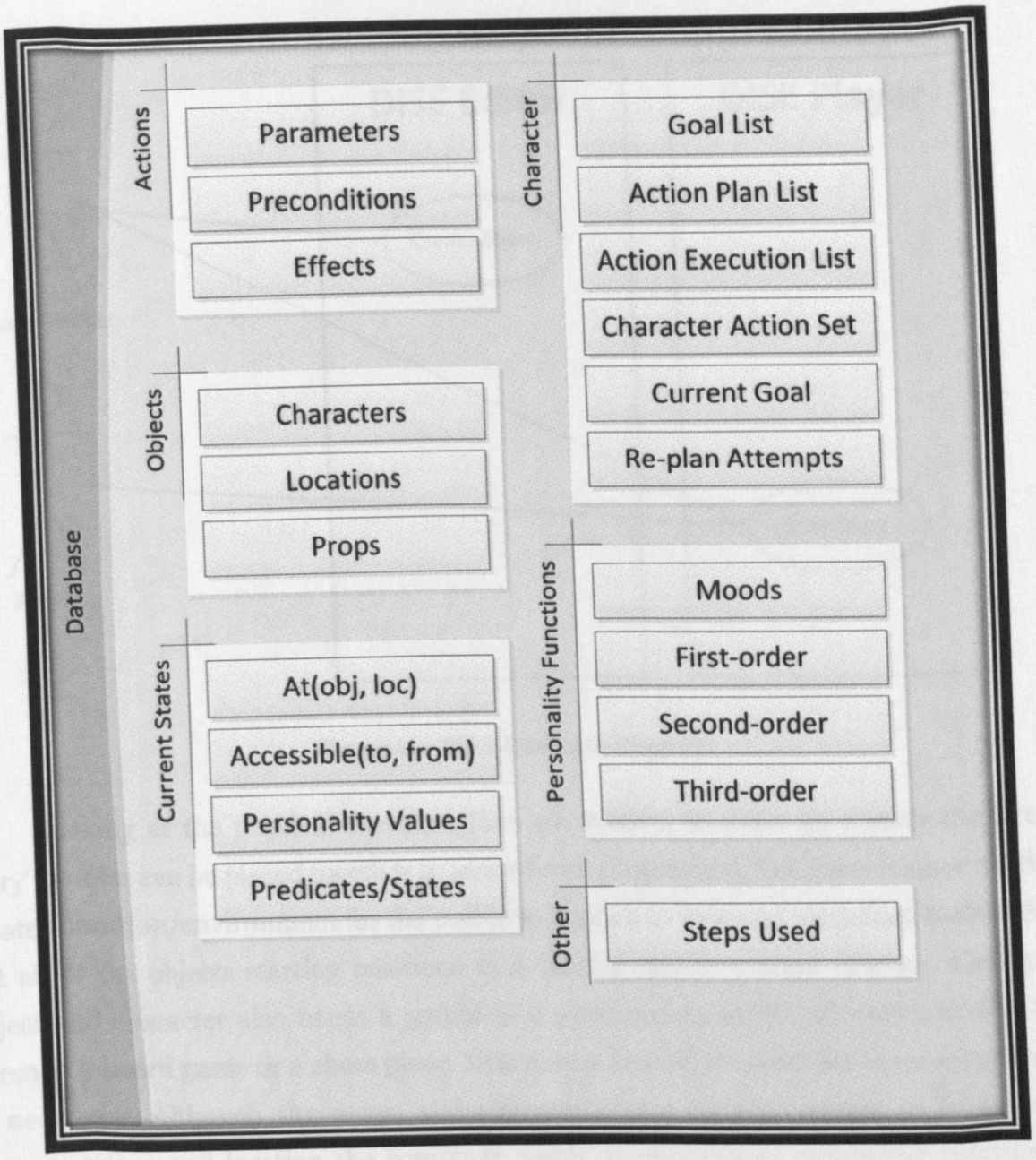


Figure 28: DISE Story Database

4.3.4 *USER CLASSES & CHARACTERISTICS*

The goals of the system can be found by looking at the users requirements. For an Interactive Storytelling framework the users are clearly defined as the storyteller (the Story Author) and the person experiencing the story (the Player). The Player has to navigate the story world and make decisions by performing actions on the things around them. On the other hand the Story Author has a more complicated role; as they have to create and manage the story assets, define what the player can do, describe the characters looks, personalities and goals and playtest to see if it works and most importantly *is enjoyable*. This means that the Player only needs to interact with the front end of the system, whilst the Story Author switches between the editor and the player components (Figure 29).

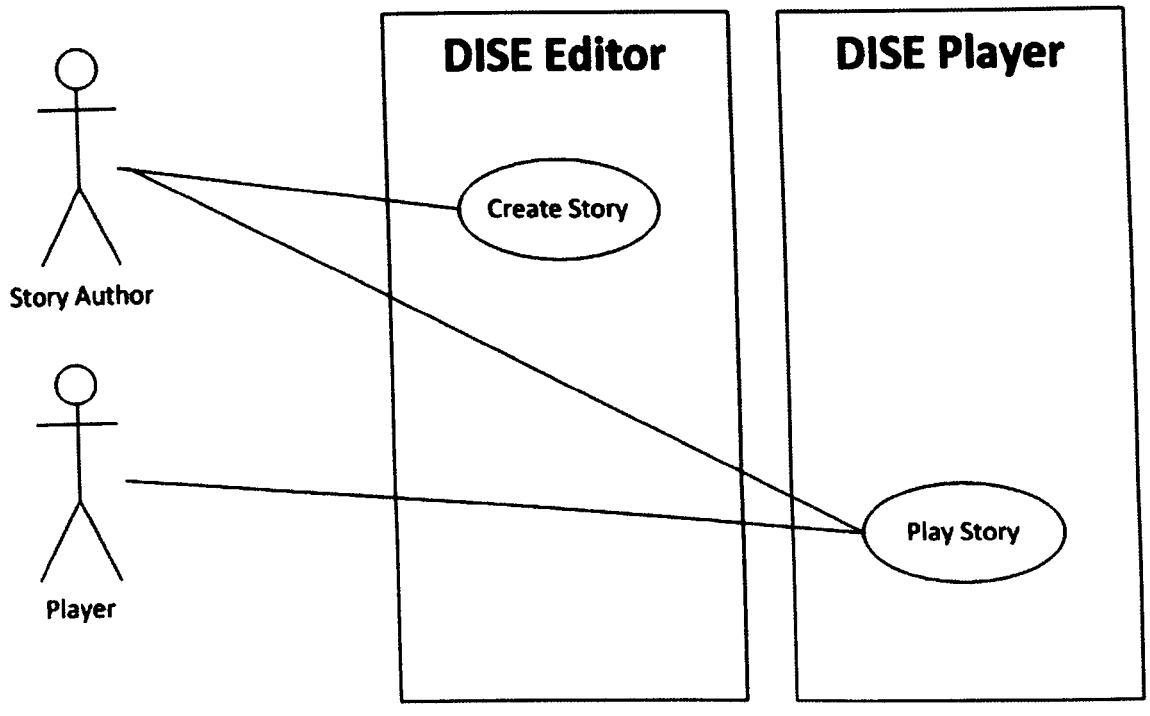


Figure 29: DISE Use-case Diagram

Looking at the physical elements that are needed to make up a story the 'create story' process can be pieced together in more detail (Figure 30). The Story Author needs to create a location/environment for the events to happen in using 3d modelling tools and lay out all of the objects starting positions in a similar way to a stage director. Each prop object and character also needs a graphical representation in the 3d world, similar to a token in a board game or a chess piece. This means that a 3d model file browser/importer is necessary. Although the props and characters now have a process to create their representation and location the computer needs to know more descriptive information about them and their properties. Props need predicate information about their current state and what they are actually used for; this data will allow the props to be coupled with the appropriate actions. The props can then be created with the following two attributes: those that can be used to do something by the actors and those that are purely aesthetic (for example background set decoration). For example the object cloak could have the predicate `iswearable(cloak)`, which would pair up with the action 'wear' that requires a prop with the wearable property set to true. Predicates could also be used to represent object states such as `isOn(deskLight)` and `isOpen(frontDoor)`. Characters require even more data such as variables to describe their moods and personalities (emotional models) and a description of their initial goals and the available actions that they are allowed to utilise to meet them.

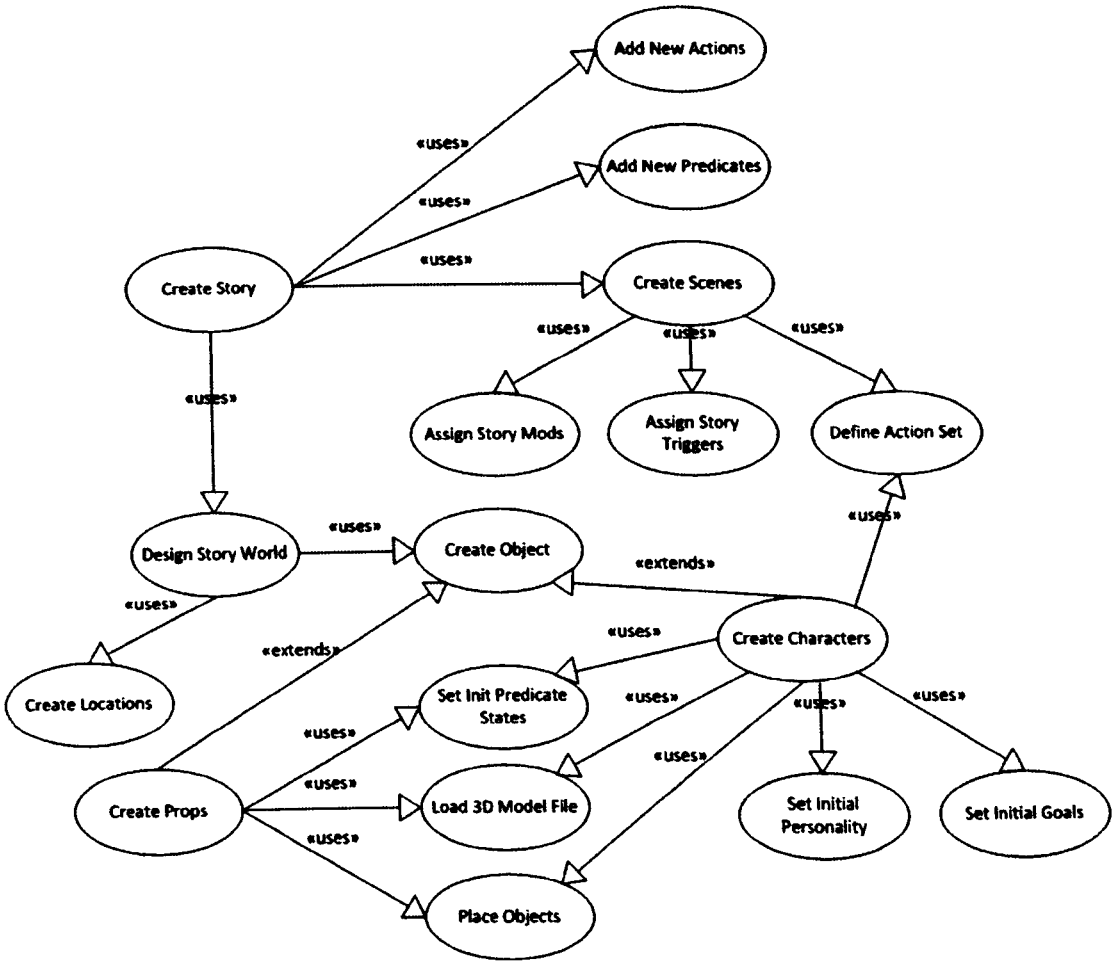


Figure 30: Create Story Expanded View

To create the story the Story Author can use the actions already available and defined in the system, but if what they need is not present or needs to be customised the an add new actions process is necessary. This also applies for the creation of new predicates. To create a story progression and direct the narrative in a particular direction the Story Author needs a create scenes process. Scenes need something to modify the current narrative taking place and make something new happen (a new event or unit of drama) and something to tell the computer when to progress the story to the next scene. This is achieved using the 'assign story mods' and 'assign story triggers' processes. Also a scene can be constrained to a particular mood or nudged in a certain direction by changing the actions a player can perform in that scene using the 'define action set' process.

Next the 'Play Story' process can be expanded to explain what the Players role is (Figure 31). The Player moves around the story world using the mouse and keyboard until they see an interactive object. The 'click objects' process allows them to select the desired object using the mouse, to perform an action on it. The 'choose actions' process then lists possible actions and may require the player to fill in further details to describe the rest of their action to understand what they want to do.

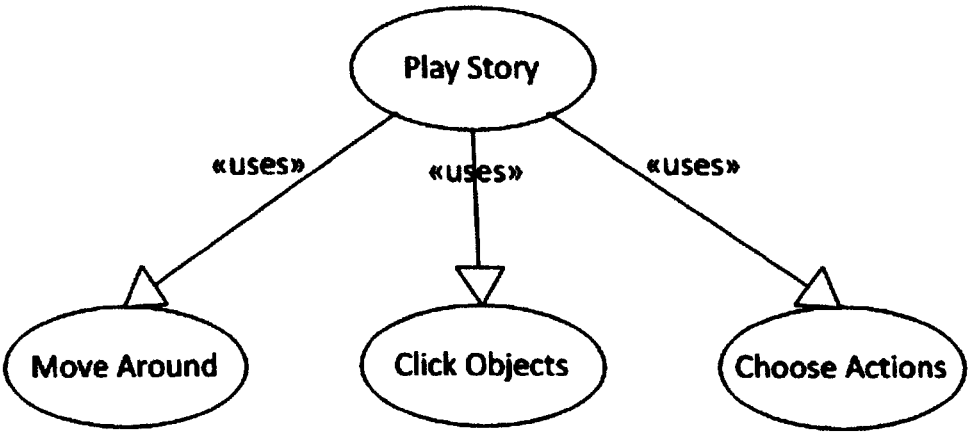


Figure 31: Play Story Expanded View

Point of view in a story can be very important and also provide context. Is the player a spectator or participant? If so, are they a single character in the story, or is the view switched between characters to reveal different perspectives and additional plot threads? This device is used in the game Call of Duty Black Ops, which cleverly reiterates a single stage in the campaign from the perspective of two characters Mason and Hudson. In the first play through Mason witnesses a character called Reznov execute an ex-Nazi scientist who defected to the Soviet Union in a key plot point. Later replaying from Hudson's perspective reveals a brainwashed Mason carrying out the deed himself with the long dead Reznov being the result of a dissociative disorder caused by the traumatic brainwashing program.

Part of the novelty of this system is to involve the player as a main character in the story instead of being a god like omnipresence that watches other characters interact.

Compton's opinion is that currently a fixed point of view works best for interactive narrative, "The day may come when changing tastes and changing visions mean [videogames] begin to follow mainstream epic fantasy into splitting the narrative. Until that time, however, it is incumbent on game designers to do everything they can to give the player an engaging journey from the single point of view—a world that comes alive just as much for the player character as it does for the player" (Compton, 2010).

Alongside a fixed viewpoint we will reiterate the player's central role in the world by assigning them a first person camera view. This means that the player sees the world through the eyes of their character, and knows only what their character knows and experiences. Game designer Ken Levine has released the majority of his games using this perspective stating that it is "the most direct way to engage the player... first person allows players to embody the on-screen character more effectively, which ultimately makes for a more engaging experience" (Pakinkis, 2011).

4.4 DISE ARCHITECTURE OVERVIEW

DISE will be made up from the following components and arranged as illustrated in Figure 13 (below). The main components of DISE as illustrated in the diagram are: the Game Engine, World Fact Database, Player Action Engine, Character Engine, PDDL Planner, Story Manager and Story Editor; and are expanded on in the following sections. There are also sections covering the story data classes that are used by these systems and the procedural content generation found in the world editor.

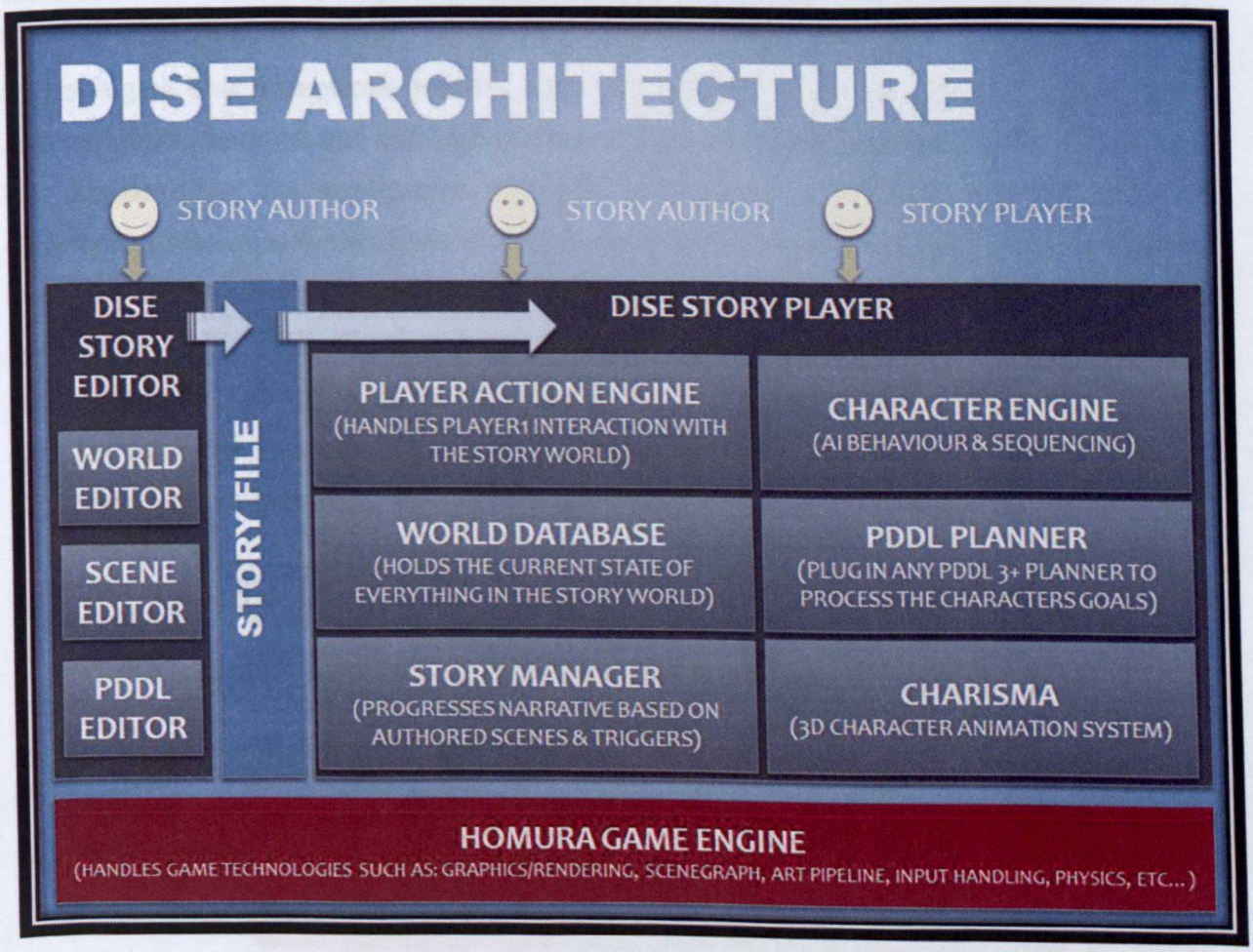


Figure 32: DISE Architecture

For the game engine and implementation development aspect we will use our Homura Game Engine and development Framework (El Rhalibi, et al., Homura: A Step Further Toward 3D Java Game Development Support, 2008) (El Rhalibi, et al., 3D Java Web-Based Games Development and Deployment, 2009) (Carter, Cooper, Dennett, & Sabri, 2008). We created the Homura project's game development framework to provide an Open Source (LGPL-Licensed) API for the creation of Java and Open GL based hardware-accelerated 3D games applications, which support cross platform, cross-browser deployment using Java Web Start (JWS) and Next-generation Applet technologies. Our framework bundles together several example applications and technical demos, which demonstrate and explain how to implement common games functionality in our applications; An application template, which acts as a great starting point for developing research applications and Homura related games; The APIs of both Homura and the key open-source projects it builds upon including the Java Monkey Engine scenegraph API, jME Physics Library, MD5 Model Importer, GBUI User Interface Libraries and many more; External Tools for the creation of Font Assets, Particle Effects and Levels for the games. Figure 33 shows the layout of the Homura Architecture and these primary components.

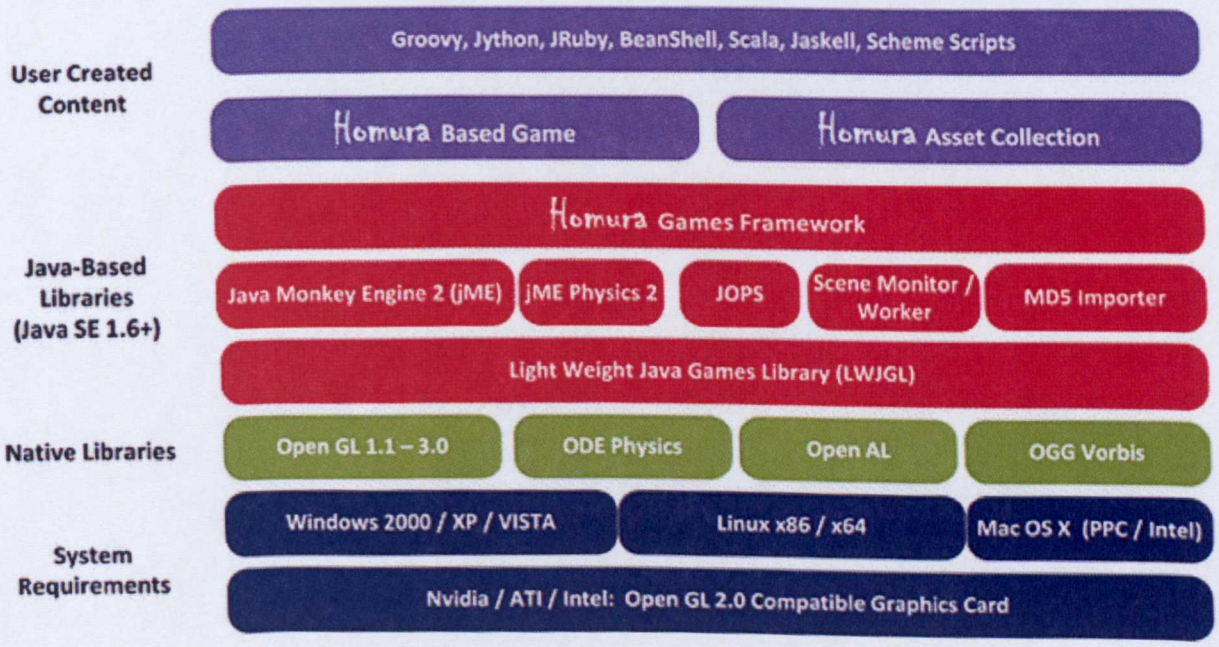


Figure 33: Logical Architecture of a Homura-based Game

We have chosen Java as the primary language within our project. By using the same implementation language and runtime for all the game development systems (i.e. engine, IDE and games) we improve the amount of compatibility and interoperability between the various sub-components we are developing. For instance, by having the engine and IDE both developed in Java, we allow the engine to run inside the IDE and allow the user to

introspect upon various parts of the engine, and debug the games during execution. This also aids multi-platform execution and reduces the overall complexity of game development in Java.

The IDE enables the development of 3D games for a variety of platforms, and the engine, include all the libraries required to render and execute 3D games. Homura provides an integrated solution for Java based 3D games, offering, Java editors, compiler, and virtual machines, as well as an extended eclipse based IDE interface. The IDE features a game spatial editor, positional editor, and a series of wizards facilitating the development of the game logic, and the management of the game assets. Different games are being developed and will be integrated into the Homura IDE, and run with the engine to experiment with the several components of the IDE, including an on-the-fly debugging facility.

The class diagram (Figure 34) below shows our main application and the example base it extends. Using this base app we can rapidly create our implementations and visualise our ideas. As shown from the example base we have created the DISE main class which loads in the main DISE components, such as the Character Engine, Story Manager, Player Action Engine and Fact Database.

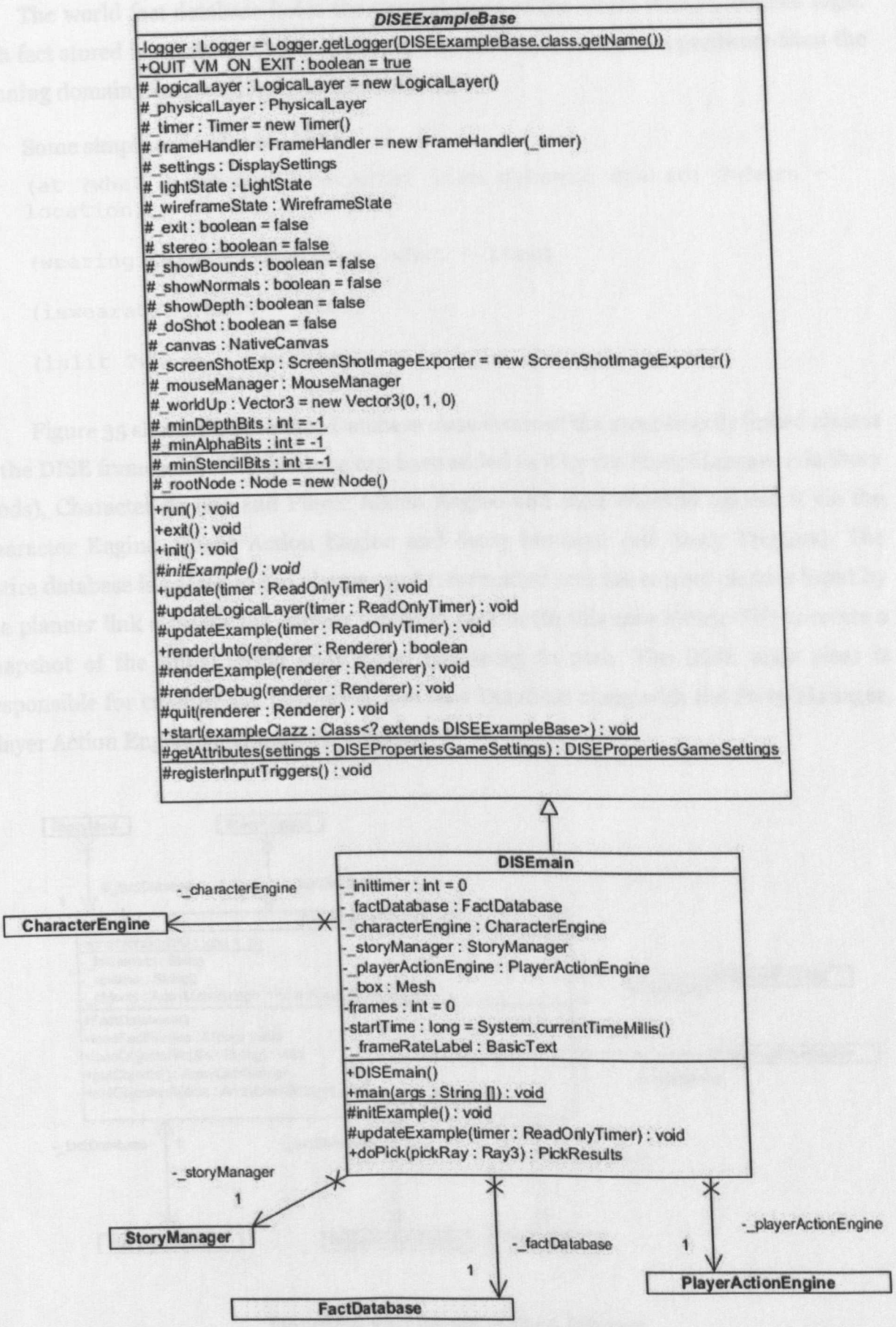


Figure 34: Game Engine Base Class Diagram

The world fact database holds the current state of the world using predicate logic. Each fact stored is a statement that is true in the world and contains a predicate from the planning domain along with its associated objects.

Some simple examples would be:

```
(at ?what - (either character item dynamic static) ?where -
location)

(wearing ?who - character ?what - item)

(iswearable ?what - item)

(islit ?where - location)
```

Figure 35 shows that the Fact Database class is one of the most heavily linked classes in the DISE framework. The database can have added to it by the Story Manager (via Story Mods), Character Engine and Player Action Engine and facts checked against it via the Character Engine, Player Action Engine and Story Manager (via Story Triggers). The entire database is passed to the planner and reformatted into the correct planner input by the planner link class for the current planning system (in this case Metric-FF) to create a snapshot of the initial world state when beginning to plan. The DISE main class is responsible for creating the instance of the Fact Database along with the Story Manager, Player Action Engine and Character Engine.

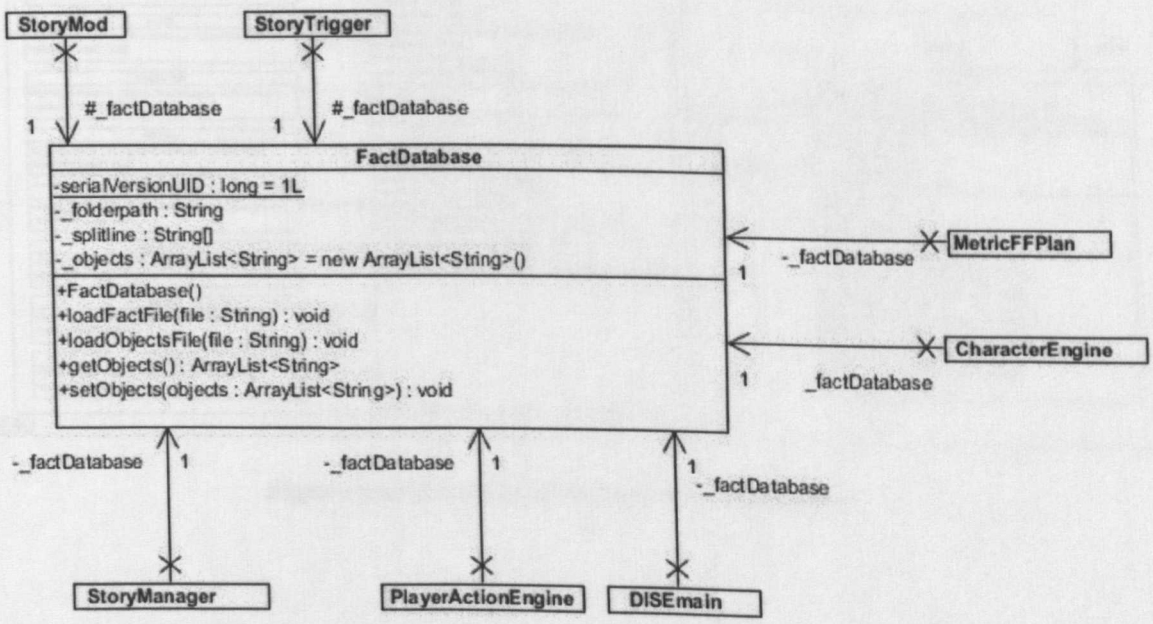


Figure 35: Fact Database Class Diagram

The story data package consists of all the information needed to create the story and the classes to hold that information.

Story Data includes: Objects (such as characters, locations, items, dynamic and static), Actions, Predicates, Scenes, the Fact Database, Goals, NPCs, StoryMods, Story Triggers and Collada models.

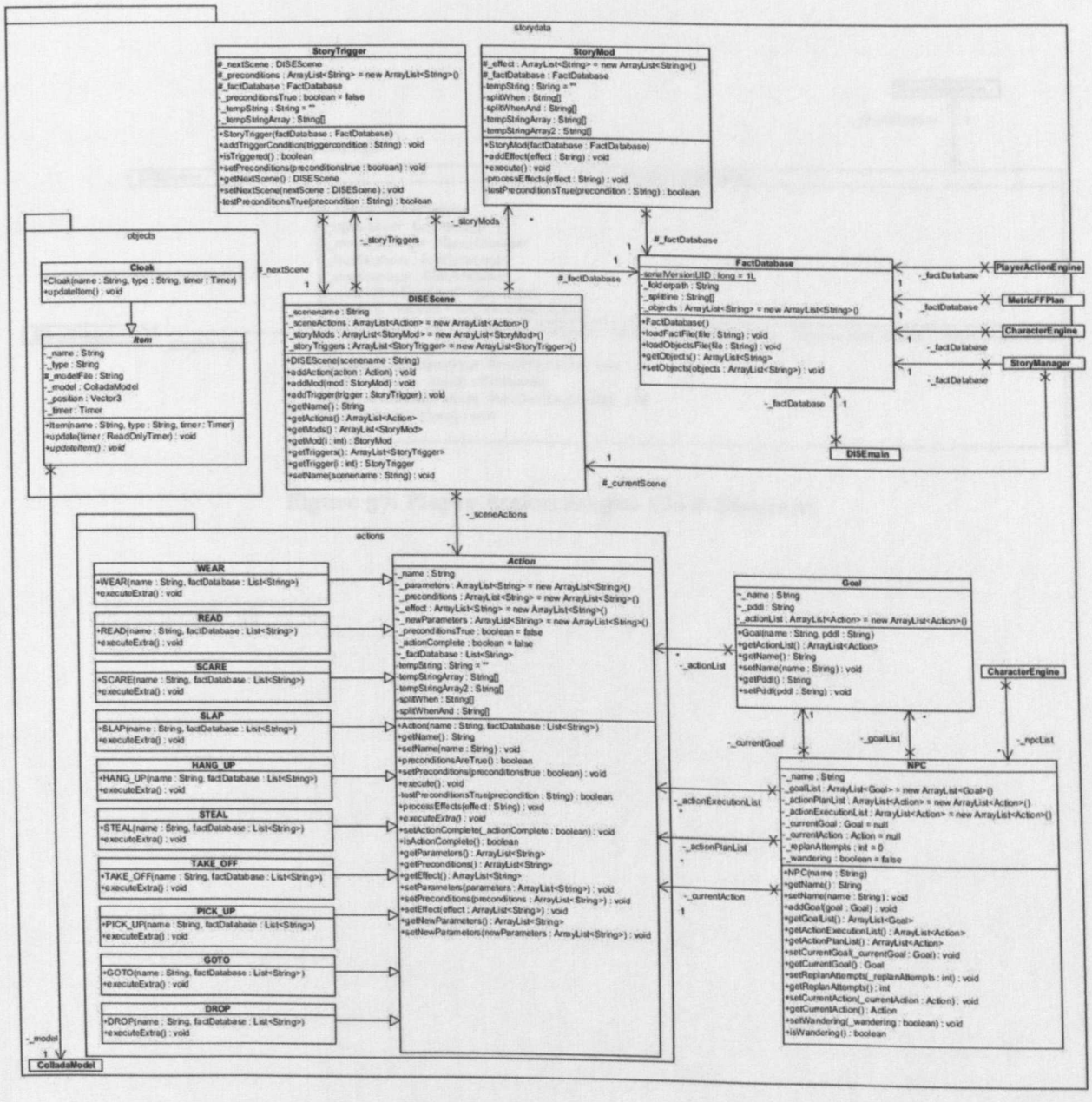


Figure 36: Story Data Package Class Diagram

The player action engine is the self-contained user interface for DISE. It allows the player to move around the 3d world and perform actions by selecting objects using the mouse to click on them. The Player Action Engine is created in the Main class and links to the Fact Database in a similar way to the Character Engine in order to check action preconditions and execute their effects by adding and removing facts.

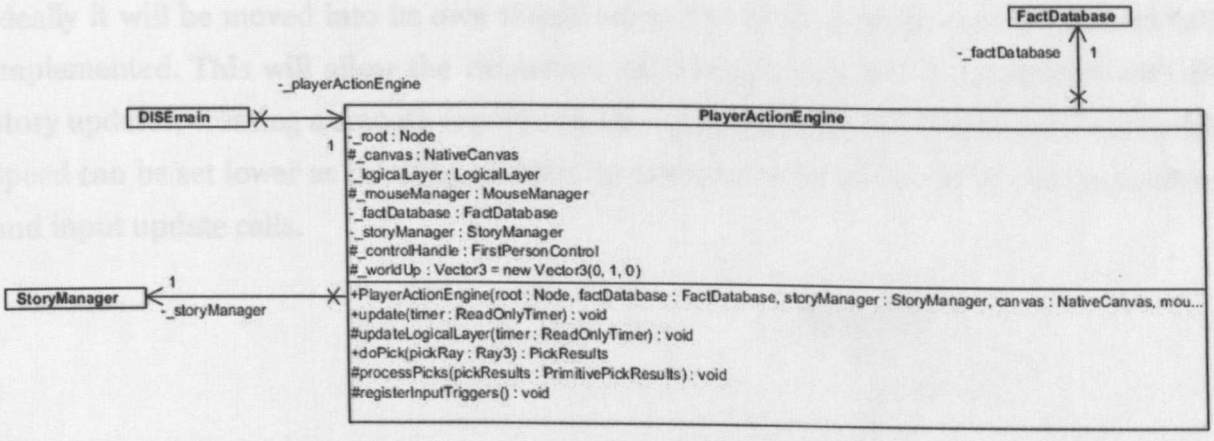


Figure 37: Player Action Engine Class Diagram

The character engine is the 'brain' of each computer controlled non-player character (or NPC) and also the sequencer of these characters (in a similar manner to a multi-agent system). The characters are listed in order of their importance and role to the current situation and are given a turn each in sequence which can result in them being instructed to carry out an action, decide what to do next using planning and re-planning and to wander (in a pre-set behaviour pattern) if they have no current goal to achieve.

In our prototype the Character Engine is updated in the main update loop, but ideally it will be moved into its own thread when the full system is more stable and fully implemented. This will allow the characters thinking process to run in parallel with the story updates, creating a smooth experience for the player. Also the step time/clock update speed can be set lower as thinking updates do not need to be as fast as the 60fps renderer and input update calls.

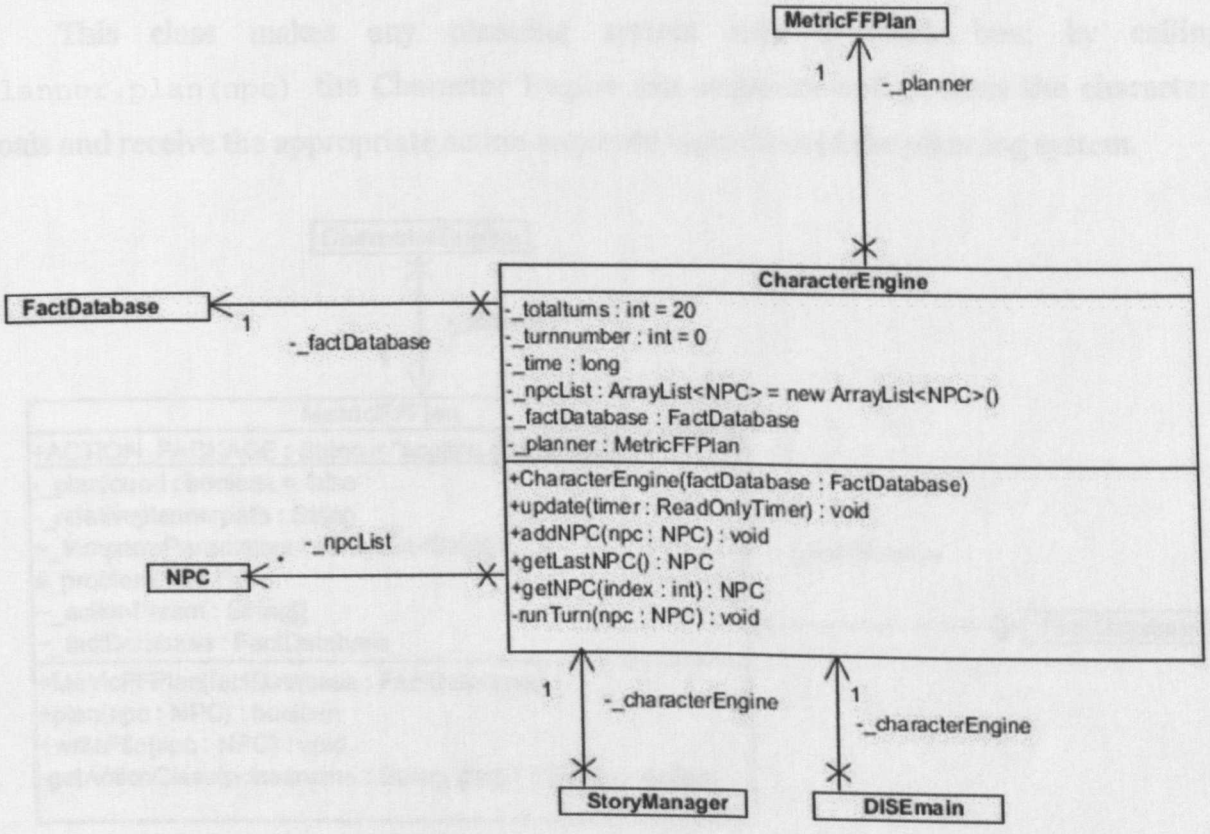


Figure 38: Character Engine Class Diagram

The planner class is designed to interface with a particular planner and gets called for each characters planning phase when their turn dictates it. Our goal for DISE is to have interchangeable planners, so comparisons can be made and the most efficient planner can be switched in.

The job of the interface class (Figure 39) is to execute the relative planner and pass in the current PDDL data from the Fact Database in the format it requires. In this case two files: `'problem.pddl'` and `'domain.pddl'`.

It then needs to process and format the planners output (in this case from the console) to pass the character a list of actions which match the available classes for each action in the action dictionary to execute in order to complete their goal. The planner interfaces also need to know when the planner has failed and return the plan found equals false Boolean.

This class makes any planning system into a black box; by calling `planner.plan(npc)` the Character Engine can sequence and process the characters goals and receive the appropriate action sequence regardless of the planning system.

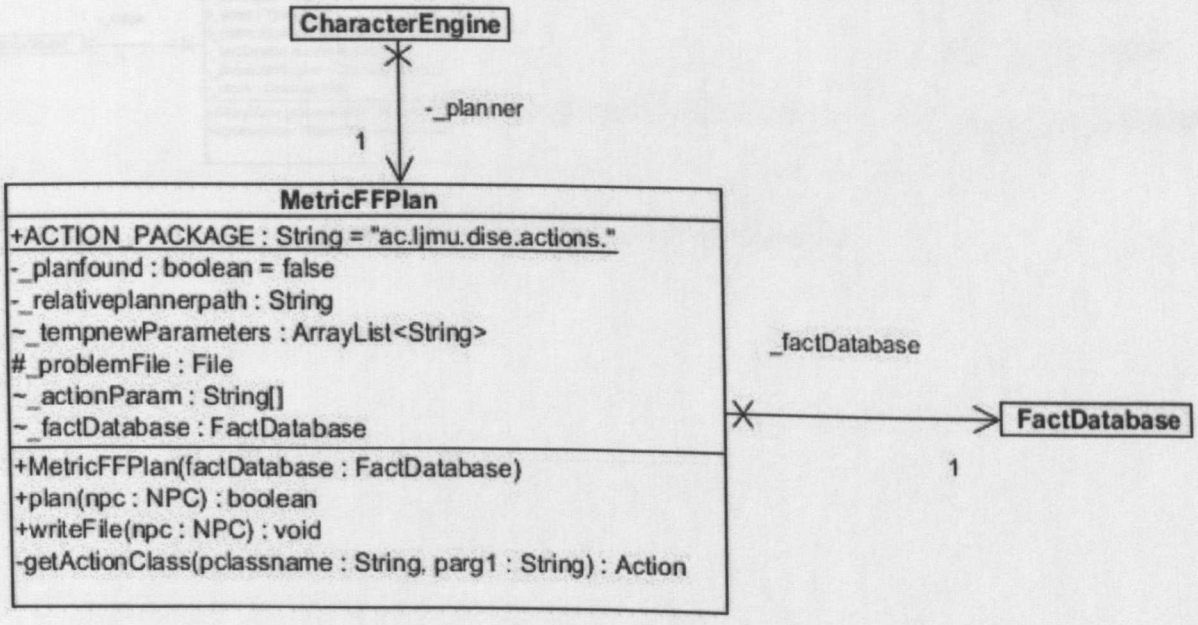


Figure 39: Planner Interface Class Diagram

The Story Managers job is to both load the story scene structure data and world data and assets and to monitor the story Fact Database, triggering narrative changes when certain values appear. The Story Manager is called by the main game loop and is the story initialiser and updater. First it loads the scene data then handles the initialisation of the story objects such as characters, locations and props, including their model and animation files. Figure 40 below shows that the Story Manager class is created in the main class and links to the Character Engine (which allows it to create and remove characters from the story) and the Fact Database (which allows it to check, add and remove facts, thus manipulating the story). It enforces the changes programmed in the current scene by the story author and also switches between scenes using the pre-coded triggers and is the main system to discreetly progress the narrative in the directions the author desired, along with processing the initial setup of the story from the saved story file and updating animation systems for in game Collada model objects.

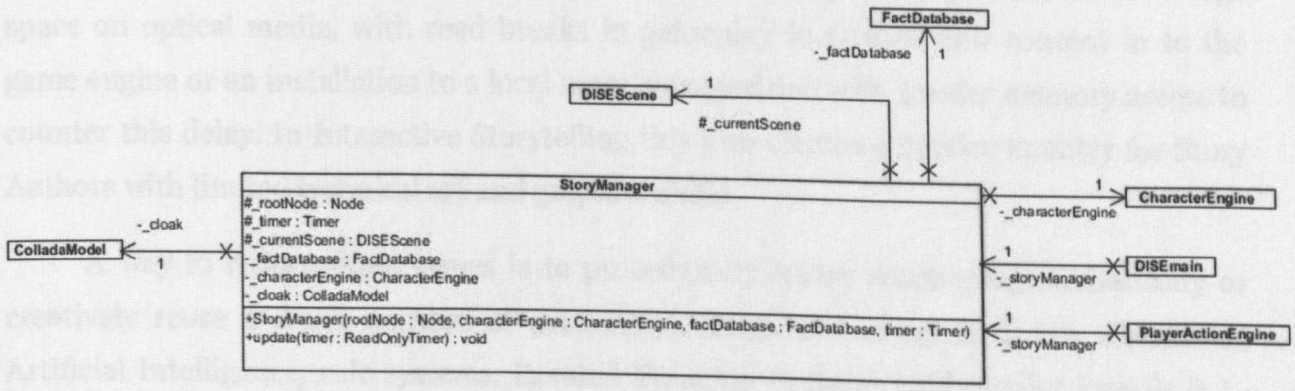


Figure 40: Story Manager Class Diagram

4.4.8 PROCEDURAL CONTENT CREATION

Procedural methods are important and show great potential in games and interactive storytelling but are an underused solution to manual content creation. Limitations to these methods include the lack of control of the output due to its random nature and the absence of integrated solutions, although more recent systems increasingly address these issues, they are rarely used to complement Interactive Storytelling. This section describes procedural methods applied to terrain modelling, including the variable realism of their output, performance and control users can exercise over the procedure to easily create stylised worlds for new stories.

As gaming machines become more and more powerful the average development time has significantly increased. A large portion of this time is used to create sprawling levels with highly detailed assets, requiring game development teams to have over a hundred full-time people, including not only dozens of programmers but also equal numbers of artists and level designers. These assets also require large amounts of storage space on optical media, with read breaks in gameplay to stream new content in to the game engine or an installation to a local mass storage drive with greater memory access to counter this delay. In Interactive Storytelling this also creates a barrier to entry for Story Authors with limited technical art and graphics skills.

A way to reduce these issues is to procedurally create assets programmatically or creatively reuse a small number of predefined assets and using recursive algorithms, Artificial Intelligence, rule systems, Iterated Function Systems (self-similar fractals & L-Systems), noise generation (Perlin), random or pseudo-random processes and shape transformations in a manner such as to avoid obvious pattern repetition.

One of the goals of DISE is to create a system that can procedurally generate content for a game level, using either random constraints or those set by a user before the generator is started. The final level should then be exportable to a generic file format, which could be loaded in at a later time.

4.4.9 EDITORS

A DISE story is made up from many aspects which would be complicated to create from scratch, so a selection of editors are needed, each one focusing on a specific content area. A story is made up from the story data, game engine data and graphical data. The DISE editors consist of:

World Editor – To create the physical environment, stages or sets where the story takes place. These are divided up into manual content creation and procedurally generated content, the former being a blocks world editor for exterior settings, a

room tiles editor for interior settings, and the latter being a city building generator and a blocks world terrain generator.

Character Editor – used to create new characters, represented by 3d model and animation files. They are also given names, start positions, default wandering behaviours, personality model settings and initial goals using the editor.

Charisma – this allows the Story Author to create complex character facial animation sequences using keyframes synchronised with voice audio.

Story Data Editors –

- Actions – used to create new action and assign parameters, preconditions and effects.
- Predicates – used to create new predicates made of parameters and types which return true.
- Initial states – used to setup the state of the world and everything in it when the story starts.
- Objects/Props Editor – Used to load in and position model and animation files that represent the objects/props in the scene and assign them physical properties, names and types.

These editors will consist of a selection of menu based panels for data entry and custom GUIs overlaid onto the HUD and around the 3d world with specific mouse and keyboard controls appropriate for each task.

4.5 CHAPTER SUMMARY

In this chapter we introduced our interactive storytelling framework called 'DISE'. We mapped out the main requirements and specifications and decided on some key features. Although some features can be found in other storytelling research, our design would be the first to include all of the following in one comprehensive framework:

- The player is a main character in the story and can move and interact freely with the world to make decisions, which have an effect on the narrative.
- Full 3D graphics and animations.
- Flexibility to create stories in multiple genres which are also not bound to one strict narrative theory for their discourse.
- Intelligent non-player characters which have their own personal goals and personalities.
- A Story Manager to constrain and progress the narrative in the ways that the author defined depending on certain events.
- Editing tools to allow new story content to be created by writers not just programmers.
- Procedurally generated environments which can be controlled with high level constraints.
- Character facial animation system with a key-frame editor.
- The ability to run cross platform and via web deployment.
- Modular framework, which can be expanded, upgraded and improved, for example: by switching the planner out for a newer version or a different planning system/algorithm/heuristic altogether.

The system was then mapped out using Unified Modelling Language (UML) including the class diagrams for each engine component in the system architecture diagram (Figure 32).

The next chapter 'DISE IMPLEMENTATION' describes how we implemented the various test applications that make up the DISE framework, starting with an overview and then explaining how each system or data class works in detail. The chapter includes sub-sections for the Game Engine, World Fact Database, Story Data, Story Manager, Planning System, Player Action Engine and Character Engine. These sections explain how each individual part was created and how they interact with the user and the rest of the framework's components.

> *"The stories we tell reflect and determine how we think about ourselves and one another. (Murray, 2004)*

5 DISE IMPLEMENTATION

5.1 DISE ARCHITECTURE OVERVIEW

The high level DISE architecture was described in section 4.4 'DISE Architecture Overview'. In this section we will explain the inner systems and their data models in more detail and will also include some diagrams and examples of the DISE code base. The basic structure of DISE was split into player and editor components then further categorised into smaller interlinked sub-systems, which each have a specific role (Figure 41).

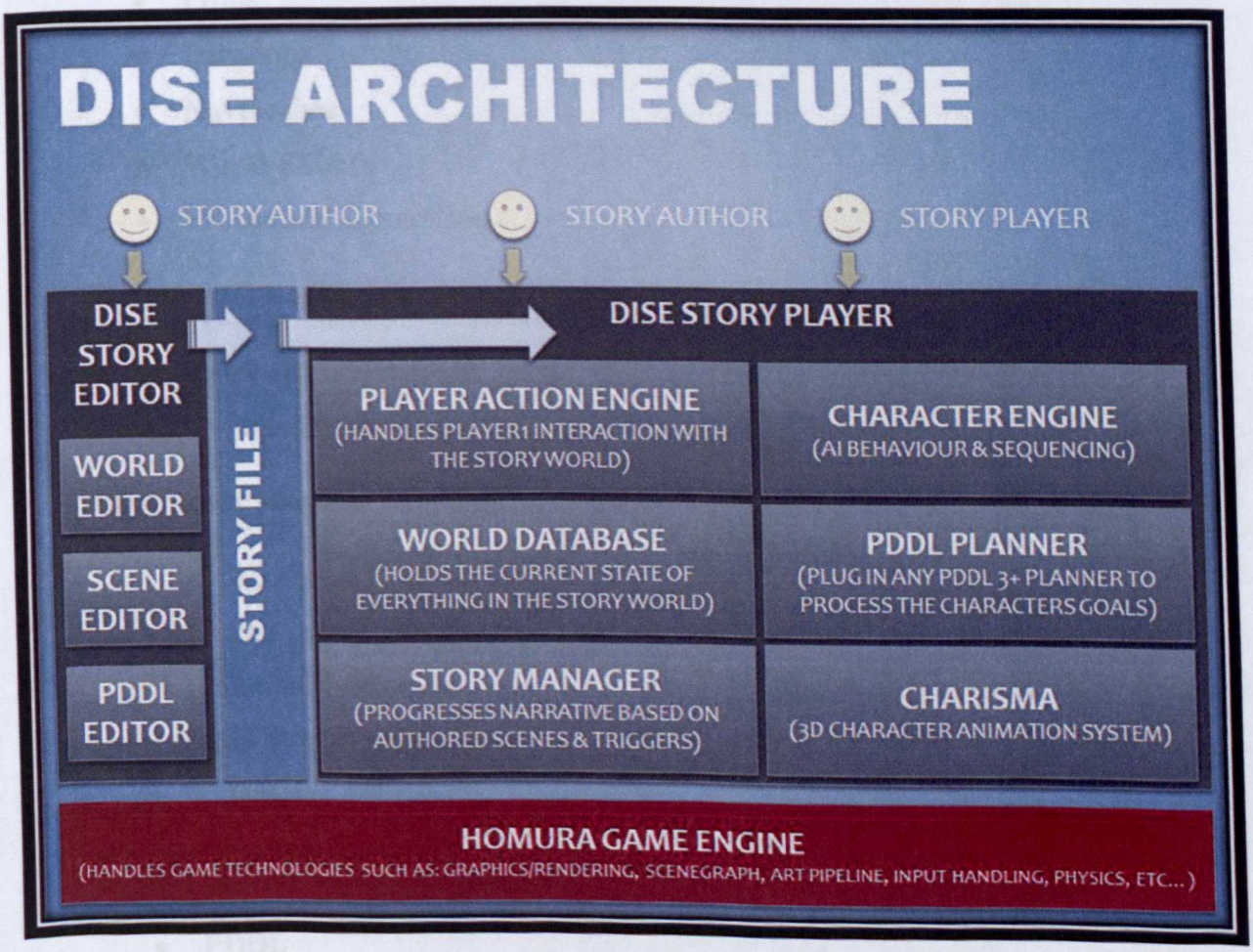


Figure 41: DISE Architecture

These sections can be divided further into the following categories that also match the following sections in this chapter:

Game Engine

- Homura

Story Data

- Types
- Verb Dictionary
- Initial States
- Actions
- Action Grammar
- Parameters, Preconditions & Effects
- Action Editor
- Props
- Locations
- Navmeshes
- World Editors
- Procedural Generation

Player Action Engine

- Player Interface & UI
- Inverse Parser

Character Engine

- Goal, Action & Execution Lists
- Turns & Sequencing
- Personality Models
- New Goal Generation
- Character Animation

Planner

- PDDL

Story Manager

- Scenes
- Story Mods
- Story Triggers

Figure 42: DISE Class Diagram

5.2 GAME ENGINE

5.2.1 HOMURA GAME ENGINE

We developed the procedural content and physics based editor test applications using the Homura Game Engine, another research project we undertook (El Rhalibi, et al., Homura: A Step Further Toward 3D Java Game Development Support, 2008). The Homura project's development framework provides an Open Source API for the creation of Java and Open GL based hardware-accelerated 3D games applications, which support cross platform, cross-browser deployment using Java Web Start (JWS) and Next-generation Applet technologies. The framework bundles together several example applications and technical demos, which demonstrate how to implement common games functionality in your application; An application template, which acts as a great starting point for development your own Homura related games; The APIs of both Homura and the key open-source projects it builds upon including the Java Monkey Engine scene graph API, jME Physics Library, MD5 Model Importer, GBUI User Interface Libraries and many more; External Tools for the creation of Font Assets, Particle Effects and Levels for the games.

The lowermost layer of the architectural stack is the System layer. Homura is a cross-platform framework and will run on Windows, Linux, Mac OS X, with the requirement of an OpenGL 1.4+ compatible graphics card.

The second layer is the Native Library Layer. Homura is coded in Java, but utilises native, platform-specific libraries for the key sub-systems. This provides the best combination of performance and feature support, allowing hardware-accelerated rendering and audio to be utilised. Homura relies on the native versions of OpenGL for rendering support, Open Dynamic Engine (ODE) for Physics simulation, OpenAL for Audio support and OGG Vorbis for open source audio format support. Java interfaces with these libraries using the Java Native Interface (JNI).

The Homura Framework comprises the uppermost layer of the API and is programmed exclusively in Java. All libraries directly referenced by Homura are also Java based, with these libraries handling the calls to the Native libraries. This approach was chosen because these existing libraries are already established and have been optimised to handle the native calls in the most efficient way, whereas Homura is primarily concerned with the high-level architecture of a games application.

Homura utilises the Java Monkey Engine (jME) to provide rendering and input handling functionality. Programmed entirely in Java, jME uses the Light Weight Java Games Library (LWJGL) as its low-level OpenGL-based rendering sub-system. The

primary function of LWJGL is to act as a Java binding to OpenGL by mirroring the interface of the C-Based OpenGL library with a Java version of each function. For example, OpenGL's `glBegin()` is adapted as `GL11.glBegin()` in LWJGL. The LWJGL function will then utilise Java's JNI system to call the native version `of glBegin()`, and uses Java's NIO system to pass information between OpenGL and LWJGL as Byte Buffers. jME provides a high performance scene-graph based graphics API. The scene-graph allows the organization of 3D geometry into a tree-like structure where a parent node can contain any number of children nodes, but a child node must contain only a single parent. The nodes are organized spatially so that whole branches of the graph can be culled. This allows for complex scenes to be rendered quickly, as typically, most of the scene is not visible at any one time. The scenegraph's leaf nodes consist of the geometry that will be rendered to the display. jME is an open-source technology which, over the last five years, has matured into a feature-rich system which is one of the most performant graphical implementations in Java for 3D applications. Homura also integrates jME's 3D Audio support.

The audio sub-system again relies on LWJGL to provide the native bridge to the OpenAL audio library, whilst using the open-source OGG Vorbis codec as the media format for audio files. Homura also utilises a jME sub-project, jME Physics 2, to provide the Physics simulation functionality of the framework. jME Physics integrates tightly with the jME scene-graph by virtue of its Physics object classes inheriting from the jME scene-graph classes. jME Physics uses the concept of Static and Dynamic node types, Static nodes are nodes that are not affected by physics, but other objects still can react physically to them (e.g. a wall), Dynamic nodes can be affected by forces and mass such as gravity and collisions with other physics objects (e.g. modelling a bouncing ball colliding with the static wall). JNI is used to bridge jME Physics with ODE to provide the low-level physics functionality. Homura also integrates with the Java Open Particle System (JOPS), a framework which allows the creation of advanced particle effects (Smoke plumes, explosions, fireworks etc.) designed for LWJGL. This has been integrated into Homura by incorporating the JOPS file type into the Homura asset management system and encapsulating the particle generators as a specialised scene-graph node called a JOPSNode allowing for easy incorporation within a scene.

The framework composites a large set of disparate components into a single system, allowing a game to be easily built on top of the Homura system through linkage with the project's binary Java Archive (JAR) file. Consequently, the final architectural layer is the User-Creation layer, which comprises the developed game. A game utilises the Homura base classes using OO inheritance and composition to provide the skeleton game - complete with all the aforementioned sub-systems. These classes are then implemented

with the required game logic and the user-developed content (Models, textures, particle effects, music, sound effects, backgrounds, etc.) which are stored as a Homura asset collection and loaded within the game classes using the Homura Asset Management System to construct the virtual environment which embodies the game.

Whilst the core of a Homura-based game is developed in Java, non-performance critical sections of the game (e.g. some parts of the game logic) can be implemented as Scripts. Homura supports a variety of languages such as Scala, Jython, JRuby and JavaScript (or any other JSR-223 compatible scripting engine), Scripts can easily be written to control any portion of the scene-graph (from the whole scene to a single node) and can be used for a variety of purposes such as AI, cinematics, animation control, event triggers etc.

We used the Homura 3d engine to create the DISE Story Player 5.7, World Editor 5.4.12 and Story Editor 5.4 main test applications, which are explained in greater depth in their respective sections.

5.2.2 OPERATING ENVIRONMENT

The initial operating environment for the DISE framework will be Windows 7, as this is installed on the two current test machines and is the latest and most widely used platform at our University. As the system is mainly written in Java it will be possible to reach our goal of deploying across other machines and operating systems such as Mac OSX 10 and Linux. It is also desirable to run DISE on computers that do not have the highest specs, using graceful downscaling.

The two current development and testing computers are as follows:

Machine 1: Windows 7, Intel Core 2 Quad Q6600 2.4 GHz, 4 GB RAM, NVIDIA GeForce 8500 GT 256MB DDR2 memory.

Machine 2: Windows 7 and Mac OS X 10.5 Leopard, Intel Core Duo at 2 GHz, 2 GB RAM, ATI X1600 with 256 MB GDDR3 memory.

To run planners programmed in C for Linux in a Windows environment, the 'cygwin1.dll' file needs included along with the recompiled executable in the resources 'planners' folder.

5.3 WORLD FACT DATABASE

The World Fact Database as described in section 4.4.2 holds all the information about the current state of the story world and its objects. This information is stored as a large list of objects and predicate facts, where each fact that is in the list is a true statement about the world that is written in the 'PDDL predicate' format. This format basically gives a vector of varying lengths for each predicate. Keeping strictly to this format allows easy interoperation between DISE and the PDDL based planners, with little to no conversion necessary. The facts can then be stored in an array list of strings and can be quickly checked for existence using Java array list's built in 'contains' function. This World Fact Database class is passed to many of the other classes in the DISE system, which can directly read from and add to the facts list. The read/write functionality is very useful for checking that preconditions are true and updating the effects of executed actions or Story Mods (heavily used in the Story Manager, Character Engine and Player Action Engine).

5.3.1 PREDICATES

Predicates have shared traits in human language and computer language and are an expression that is true of something, expressing a relationship, or property of an argument in a clause. Predicates are the second main part of a sentence, with the other being the subject that is modified. In PDDL predicates can have multiple subject parameters along with their corresponding types and always follow the pattern of: predicate-name, parameter, and parameter-type.

Example of the 'at' and 'have item' predicates are given below:

```
(at ?who/what - (either character item dynamic static) ?where
- location)
(haveitem ?who - character ?what - item)
```

Predicates are used to represent the state of all objects (nouns) in the story domain and are the main components in the Fact Database along with the list of objects in the story (which are used as the subject parameters for the predicates). Using these predicates to define the state of the world (by adding and removing them from the Fact Database); we can represent the initial state, goals, and actions (with preconditions and effects).

5.3.2 PREDICATE EDITOR PANEL

The Predicate Editor Panel allows new predicates to be added to the list and used in a story. New predicates can be used to describe new states for existing objects or states for entirely new objects. There are no rules as to how many predicates are used to define an object's many states, but the Story Author has to be careful that none logically conflict with each other. To help and also avoid PDDL syntax errors the editor will not allow

predicates with duplicated names to be added to the list.

New predicates can be created by typing a unique name into the large box and clicking the add button. They can also be removed using their delete (X) button. Once created, they need to have their parameters and parameter types set. This is achieved by clicking the corresponding edit button and adding a row for each parameter. Parameters are automatically named x, y, z, w, etc. to make things easier. The Author then just needs to put a tick against each type that the parameter shares. Figure 43 shows how the 'at' predicate in the previous section was created. The first predicate (who/what) has types (either character item dynamic static) and the second (where) can only be a location. When finished the save button can be clicked to save and close the popup. Defining types limits what objects can be set as the subject parameter of the predicates and also helps when defining actions, goals and initial world state in the other editor panels by narrowing down the available choices in the drop down menus.

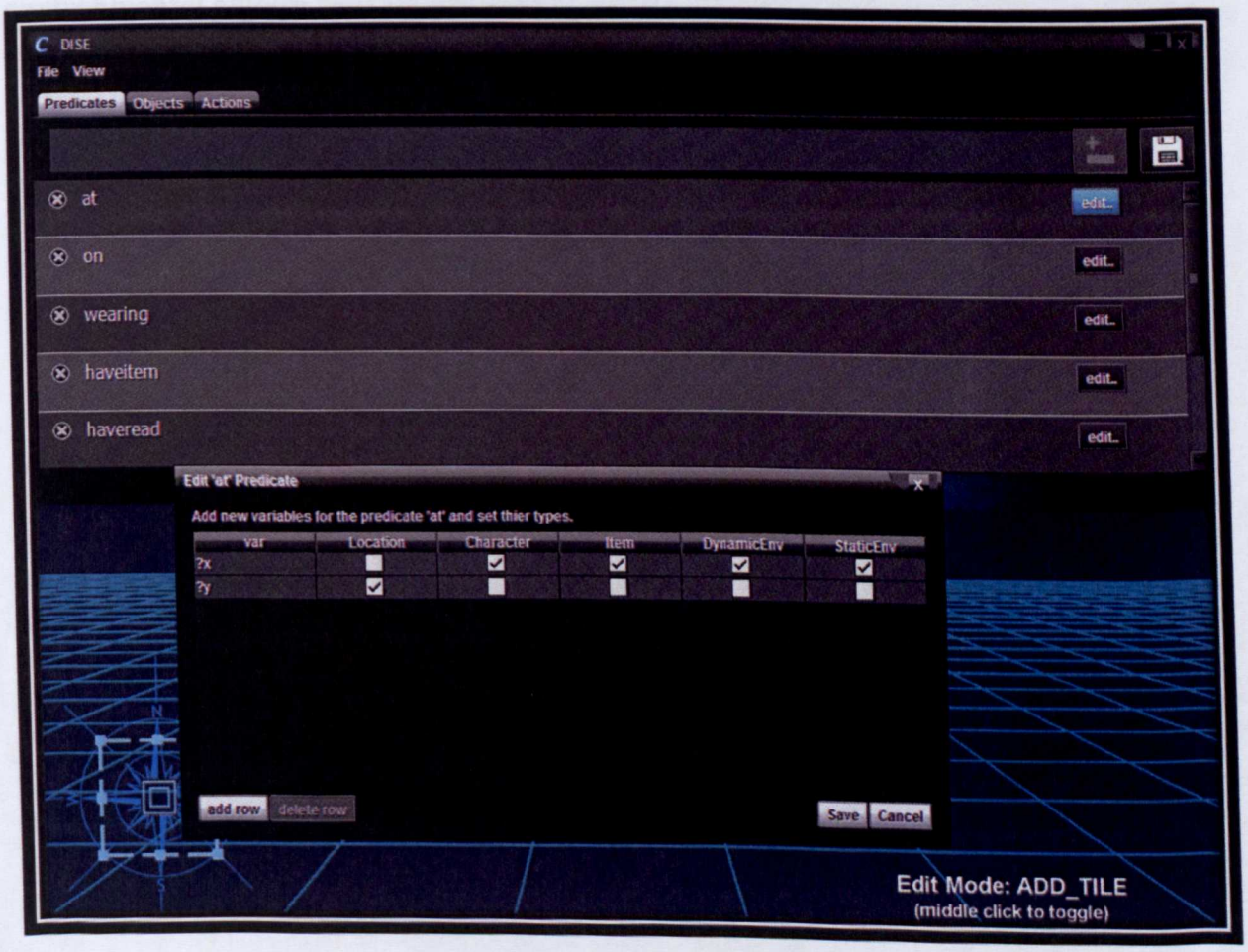


Figure 43: Predicate Edit Panel

5.4 STORY DATA

5.4.1 TAXONOMY OF THE STORY WORLD USING TYPES

The structure of the story is the most important aspect of DISE and will also dictate how the player can interact with the virtual world. The base concept uses a hierarchy of words to create taxonomy of the game world and everything in it (Figure 44). The base general instance that every object is derived from is 'word'. This is then broken down into 'verbs' for actions and 'nouns' for 'props, 'characters' and 'locations'; thus allowing a type definition for each individual object instance or a collection of things. For example 'props' are everything added to the story that is not a character or location, 'information' is treated as a prop that a character can have (if they know) and 'items' are props that can be picked up and stored in a character's inventory. The 'dynamic' environmental objects are props that cannot be taken, but have a level of interactivity, such as a door and 'static' environmental objects have no interaction apart from blocking movement using collisions. This type definition also marries well with the type hierarchies needed for our planner.

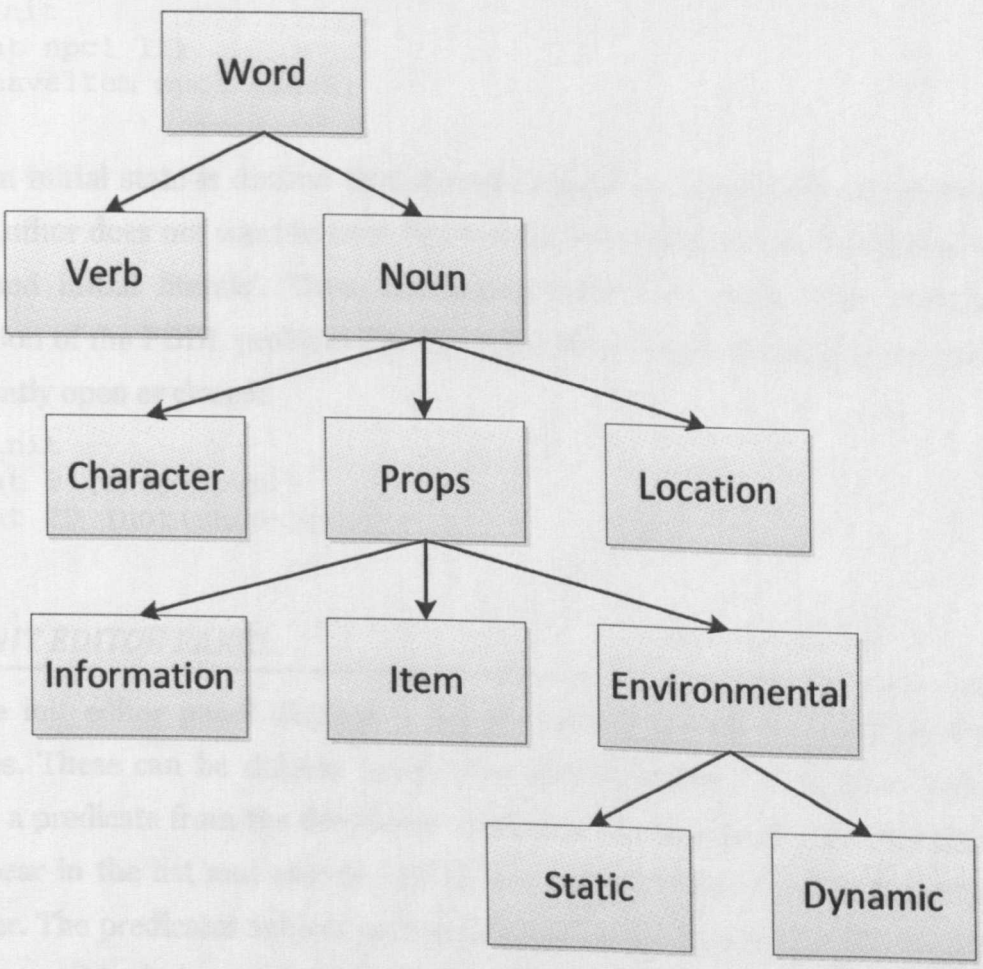


Figure 44: Type Hierarchies

5.4.2 VERB DICTIONARY

The verb dictionary is a list of all action definitions that have been saved into the storytelling system for every story so far. This is the equivalent of a master list of actions. In the final system we will include a base list that can be expanded on by creating new actions or editing existing ones using the editor and saving them to the verb dictionary. We would also like a way for these to be shared by story authors and used as a general repository for quick story building. We have given each verb its own Java class The author can scroll through all the available verbs and choose which ones they need for their story, which moves them from the master list over to the current list.

5.4.3 INITIAL STATES

Initial states are just a list of predicates (see section 5.3.1) with objects assigned to all of their parameters. Using the predicate example in section 5.3.1, we could replace the '? Parameters' and use 'at' and 'have item' to initialise npc1 as being at location 1 whilst possessing the cloak item:

```
(:init
 (at npc1 l1)
 (haveitem npc1 cloak)
)
```

If an initial state is desired that changes regularly, depending on the current time and the Author does not want to complicate their scene transitions, then initial states can have 'timed initial literals'. These are states which are given time switches in the initialisation of the PDDL problem file. For example a shop's opening hours can dictate if it is currently open or closed:

```
(:init
 (at 9 (shop-open))
 (at 20 (not(shop-open)))
)
```

5.4.4 INIT EDITOR PANEL

The init editor panel displays a list of the initial state of the story world using predicates. These can be deleted using their corresponding (x) buttons and added by choosing a predicate from the dropdown menu and clicking 'add'. The new predicate will then appear in the list and objects can be selected from the dropdown menus for each parameter. The predicates subject parameters will only list objects of the matching types, so no impossible facts can be added. For example 'have item' would only have two dropdown menus (one for each of its two parameters), containing only characters and items respectively.

5.4.5 ACTIONS

Actions describe everything characters (and player) can do in the story world. The actions are defined by verbs and can be linked to nouns to create full action phrases. Actions are made up of data the describes what variables they are concerned with, when they can be carried out, what their effects are on the story world, what reactions other characters should have to them, what animations should play for the action and Java code to update the game engines objects and the players view of the world.

5.4.6 ACTION GRAMMAR

To create the full description of an action we need to form a sentence using a set grammar that humans can understand and that the computer can interpret. To achieve this each action needs a verb to describe the action, immediately followed by an agent to carry out the action. The next part of the sentence can vary depending on the subject of the action and the objects needed to carry it out. This third word will be a noun of type character, location, prop, or dynamic object. The next word is an optional noun, only used by some actions that need to combine more objects that interact with each other. Any other nouns can also be added after this, stretching the length of the action sentence to n words, where n is the number of parameters formally defined in the action's PDDL code (see the next section for more detail on action PDDL). The last word is always the location where the character needs to be to perform this action, which allows the characters to plan their movement and path-find to the correct area.

The final action sentence format is then:

```
VERB, CHARACTER, NOUN, (OPTIONAL)NOUN, ..., LOCATION
```

A basic action could consist of only three words:

```
take-off(?who - character ?what - item)
take-off(npc1, cloak)
```

Where npc1 is a computer controlled character and cloak is an item that is currently worn by npc1.

An example of a more complex action sentence that uses five words would be:

```
hang-up(?who - character ?what - item ?hangOnWhat - dynamic
?where - location)

hang-up(npc1, cloak, hook, c1)
```

Where npc1 is a computer controlled character, cloak is an item that can be hung up, hook is an interactive (dynamic) prop which can have an item hanging on it and c1 is sub-node 1 of the cloakroom's navigation mesh (see section 5.4.13 for more info on navmeshes).

5.4.7 PARAMETERS, PRECONDITIONS AND EFFECTS

To correctly sequence and describe our story actions we will use a branch of artificial intelligence called automated planning and scheduling. The main data to describe actions is stored in the Planning Domain Definition Language (PDDL) format. PDDL is a recent attempt to standardise planning domain and problem description languages and includes STRIPS, ADL and more via extensions. This gives the most important information about the actions nature that the computer needs to understand when the action is possible, which variables it is concerned with and what its effects are.

The actions are usually made up of three parts:

- **Parameters** – Usually the 'Name' and 'Type' of an object, character, or location that are used by the action.
- **Preconditions** – Requirements that need to be met (i.e. states of the world that must equal true) to perform the action.
- **Effects** – Post conditions and modifications to the world's states after the action is carried out.

This data can be used both for the non-player character AI to plan and sequence which actions need to be performed to reach their goal and to calculate the adjusted world state after an action is completed by either the player or a computer AI agent. Later we will also explain how the parameters and preconditions can be used to select a subset of actions that fit the player's current context using inverse parsing (see section 5.7.2). The effects PDDL description is a useful tool that can be used not only to plan, but to update the world state data (the model). In order to reflect this in the game engine (the view) we also require some Java functions attached to the predicates that get called when the effects are processed. The following table (Figure 45) shows further details on how the structure of a verb file will look in the verb dictionary. Some brief examples of the verb 'give' are shown in pseudo code underneath each section heading.

This verb system can be used to represent the most important state modifying verbs mentioned in Todorov's model (Todorov, 1969) as outlined in section 3.2.7 NARRATIVE GRAMMARS. These are:

- Mask or unmask: while the situation might not actually be modified, someone is made to believe it is modified (knowledge).
- Words - calm a situation; hurt a situation (personality model).
- Physical - attack or resistance.
- Seek help - by evoking compassion or asking for advice.
- Change of location – (go to/walk).

- Exchange – (trade) through payment or giving freely (shown in the example below in the table: Figure 45). The flexibility of an Actionary gives the story author a wide scope to create any action and to even expand out of Todorov's model if they see fit.

Breakdown of an Action Verb

Name: **Name of the action**

Example:

```
Action: Give
```

Parameters: **Usually the 'Name' and 'Type' of an object, character, or place that are used by the action and in the format:**

```
?parameter_name - parameter_type
```

Example:

```
Giver - Character, Receiver - Character, Item - Thing/Object,
L - Location
```

Precondition: **Requirements that need to be met to perform the action.**

Example:

```
Giver != Receiver And Giver Has Item And At L Giver And At L
Receiver
```

Effect: **Post-condition & states after this action is carried out.**

Example:

```
Giver !Has Item And Receiver Has Item
```

Figure 45: Breakdown of an Action Verb

Using the actions panel in the editor new verbs can be created fairly easily and without knowing the exact PDDL code syntax. Figure 46 below shows the prototype action editor and the popup windows used to create and edit actions with their parameters, preconditions and effects.

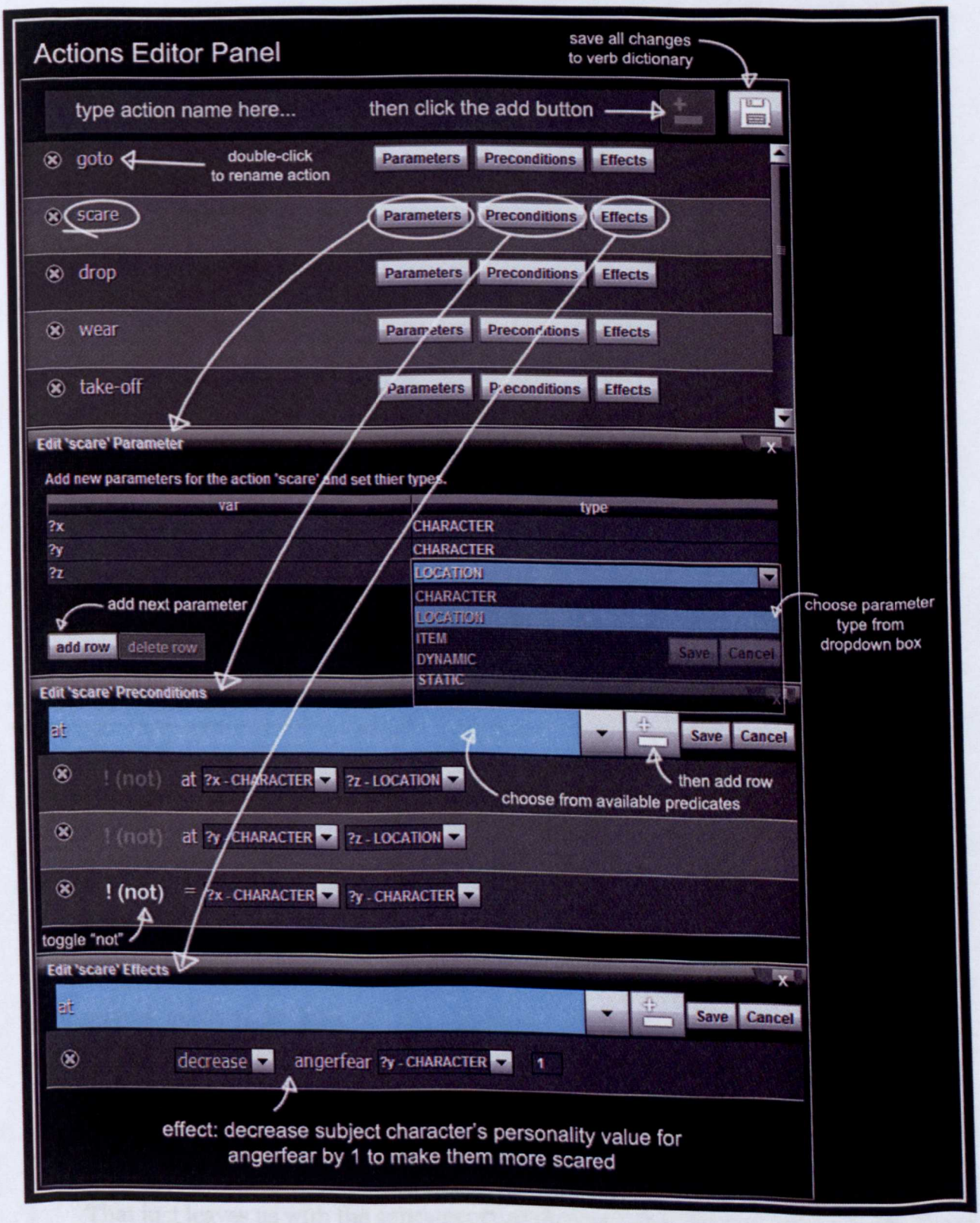


Figure 46: DISE Action Editor UI Panel

The following code is produced from the information input in the diagram above:

```
(:action scare
 :parameters (?x-character ?y-character ?z-location)
 :precondition (and (at ?x ?z)(at ?y ?z)(not(= ?x ?y)))
 :effect (and (decrease (AngerFear ?y) 1))
)
```

To create this code first the action name is typed in the top text box and is then added using the 'add action' button. It will be appended to the scrolling list of all actions below and can be re-named by double-clicking its text label and changing the text. At first the parameters, preconditions and effects will be empty so each one needs to be filled in with the appropriate data by clicking the corresponding button to open a new window. The parameters section needs a list of the variables concerned with the action and their types. The 'add row' button adds a new variable which is automatically given a name (x, y, z, w, etc.). The types can be selected using the dropdown box in the second column and selected parameters can be removed completely with the delete row button. To edit the preconditions the main dropdown is used to choose an available predicate for the particular precondition. Then pressing the add button will add the predicate to the list. The editor will automatically filter and list any parameters that are useable in the new predicate row. These can be changed to any parameter/type available to make the correct precondition test. Clicking save will save and exit the precondition popup window. The final effects can be added with the effects button, which opens up a similar window to the preconditions. Here effect predicates and their parameters/types can be defined and saved (Figure 46).

5.4.9 OBJECTS & PROPS

The Objects in PDDL are any Locations, Characters, Items, Dynamic environment object or Static environment object (see section 5.4.1 for taxonomy explanation). The object PDDL definition only needs a name for each object in the story and its type, however more information than this is needed for the story world to be drawn correctly by the game engine.

'Locations' are defined separately in the DISE world editor as the whole environment needs to be built up and positioned and its navmeshes generated for each location, so are covered later in section 5.4.12.

Also 'Characters' have much more data, such as personalities, goals, etc. so are covered later on in the character engine section (5.8).

That just leaves us with the subcategory of objects called 'Props'. Props are similar to their stage counterparts and are further broken down into 'Items' (things that can be carried in a character's inventory), 'Dynamic environment objects' (things that are usually

larger and can be interacted with, but not carried around) and 'Static environment objects' (things that form barriers but cannot be interacted with or moved). Static props only have game engine data as they are not used in the planner. Item props usually have a model file to represent them, a location and vector position and a dynamic physics setting. They can also contain their size in number of inventory blocks needed to store them and the pattern of these blocks. Dynamic prop objects have their physics set to static or dynamic (depending on what they are), a model file, location and vector position, but do not require inventory block data as they are never carried. This data can be created using the editor panel and free drop editor detailed below.

5.4.10 OBJECT EDITOR PANEL

The object PDDL definition, physics type and model file can all be easily input using this panel in the Story Editor. A new object can be created by typing its name into the top box and hitting the add button. This will create a new row in the objects list, which can be deleted or edited further. There are two drop down boxes allowing the story author to choose the type of the object and the physics type and a file browser allows the easy location and importing of Collada model files. Items can also be given an inventory block size and shape by highlighting grid squares if this is required in the story. These objects can then be positioned using the prop free drop editor mentioned in the previous section to give them a position location and their exact vector data.

Figure 47: Object Edit Panel

5.4.11 PROP FREE DROP EDITOR

A physics based editor prototype was created using the Homura Engine, which allows new props to be dropped freely into a level. The props are created using their model and texture data and then wrapped up as physics objects (see Figure 48). These objects can be given the following physical properties:

- Dynamic – these are moving objects that collide with other static and dynamic objects.
- Static – these are non-moving objects, such as a solid wall, that can collide with dynamic objects.
- Phased – these objects can pass through any object and then be set to static as above. This is useful for a column partially buried in the ground.
- None – ignore collisions for this object. This is useful for objects that are purely aesthetic, such as scenery placed out of bounds in the background of the level.

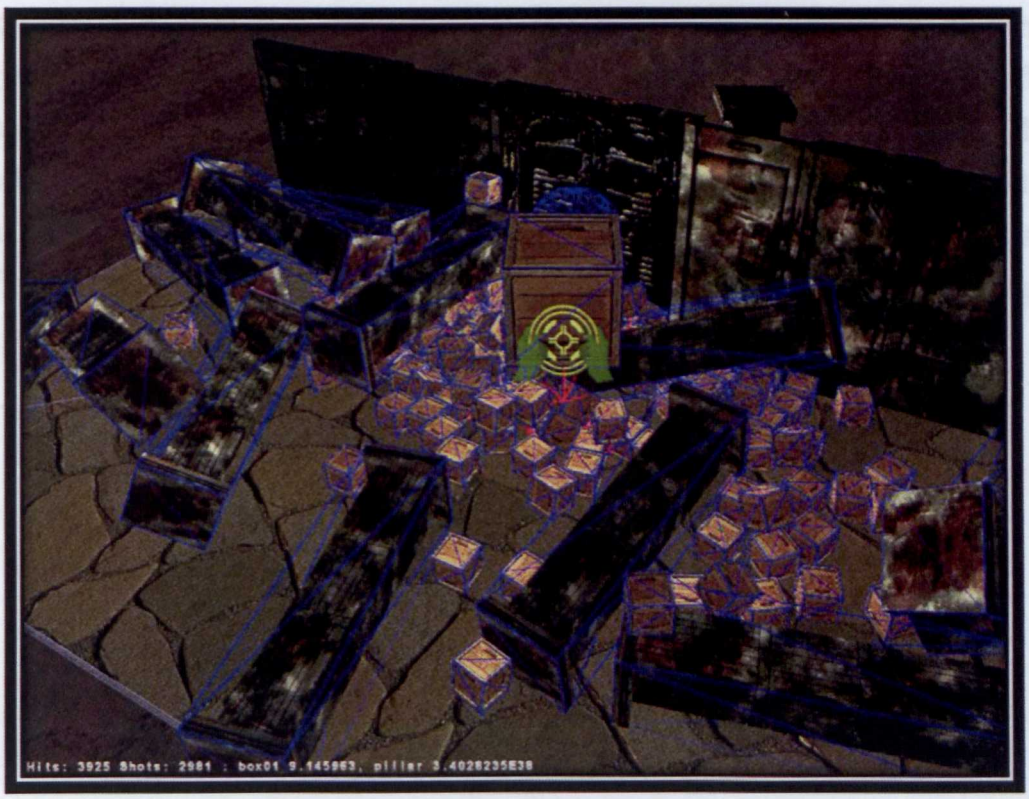


Figure 48: DISE World Editor Free Drop Mode

Dropped objects can be picked up by aiming the mouse reticule at the object and clicking. This will create a physical joint between the camera and the object, so the object will follow the user's movement. When an object is picked up extra keyboard commands allow it to be rotated in 3d and with another click re-dropped into its new position. When an existing object is selected with a mouse left click a menu will appear with the options to move or delete the chosen object (Figure 49).

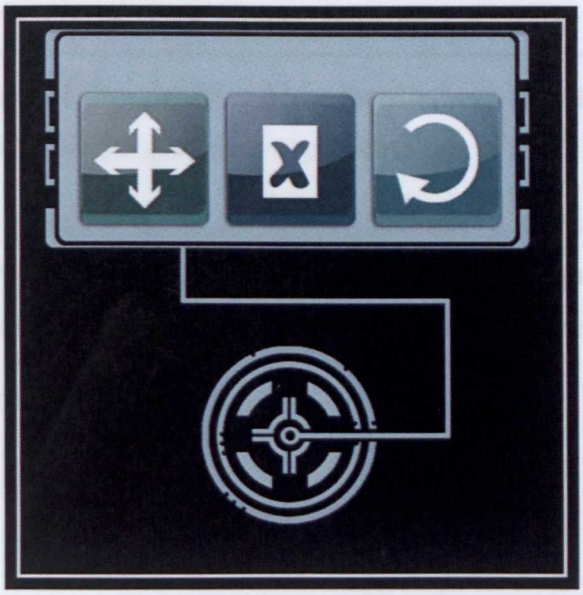


Figure 49: Editor Move/Delete Menu

If the move command is selected the cursor will change to a hand and the object should now float in front of the camera. Using the mouse to look around and the arrow keys to move, the object can now be positioned anywhere in the 3D world. After moving the object it can also be rotated to the desired direction and placed with another left-click. As some objects are affected by physics properties, they may not be able to pass through each other, when an object is dropped if it is dynamic, it may bounce around and find a natural resting place. The editor will also support new object creation, either from a list of predefined objects or allowing the user to load their own model files and editing some object properties to describe how it will behave in the story world.

Locations are areas of the virtual environment where the story takes place. Using various editors detailed below the graphical representation of these locations can

The 'Accessible' function specifies which locations a character can get to from another location and requires two parameters: x- location to (the location that can be accessed) and y - location from (the start point from which it can be accessed).

The PDDL function code is shown below:

```
(:functions
(accessible ?x - location ?y - location)
)

(:init
```

; showing that the exit is reachable from the foyer

```
(accessible exit foyer)
```

; showing that the foyer is reachable from the exit

```
(accessible foyer exit)
)
```

Using both of these statements means that the locations have bi-directional access. If only one 'accessible' function is used for this pairing of locations then a one way system is created. This will only be used in certain cases such as a turnstile gate. If this data can be created in the editor automatically or generated when the game starts, the agent planning could incorporate node-to-node navigation and even find the most cost effective route using the 'minimize' function in the PDDL 'metrics' requirement and a cost for each node traversed.

For example:

```
(:metric minimize (total-distance-travelled))

(:init (= total-distance-travelled 0))
```

Can be used with the 'goto' action, by adding an increment cost to its effect, giving:

```
(:action goto
 :parameters      (?x - character ?from - location ?to -
                  location)
 :precondition    (and (at ?x ?from)    (accessible ?from
                  ?to))
 :effect          (and
                  (at ?x ?to) (not (at ?x ?from))
                  (increase total-distance-travelled 1)
                   )
)
```

Navigation meshes (or navmeshes) are strips of convex polygons that are overlaid onto the world map to define the areas that are 'walkable' or not. The non-player characters can use these areas as waypoints to navigate the scene, usually moving between each polygon's centre points. Tozour (Tozour, 2008) lists the five benefits of using navigation meshes instead of traditional waypoint systems as:

1. Big worlds need large numbers of waypoints vs. the simplicity of navmeshes means fewer nodes and faster real-time pathfinding.
2. Waypoints force characters to take a zigzag route vs. navmeshes can use smooth spline paths as long as they remain inside the mesh's area.
3. Navmeshes allow path correction for dynamic obstacle avoidance.
4. Waypoints don't work well for different sizes of characters/vehicles due to their lack of volumetric information.
5. Navmeshes can be used for collision tests during character action animations, so extra collision ray-casting is not needed.

When structures are created or loaded into the world editors their navmeshes can be pre-generated using various tools (such as the Slick Library for tile based levels and Recast for complicated scenes). These then need to create a graph of accessible nodes so that the characters can plan a route around the world towards their goal location. One method is to find where two polygons edges join then find the centre point of the smallest edge.

In the example below (Figure 50) a navmesh has been generated for the cloak of darkness story world (see Section 6.1.1. for the full description). This building has four rooms all divided up into numbered subsections: the main Foyer (divided into 8 nodes), the Bar (divided into 9), the Cloakroom (which also has 3 nodes) the Exit (with 3 nodes). Each mesh is drawn as a green box and given a node number label and the adjacent edge's centre points are marked with small red squares. From this navmesh the data needed for the characters plan generation can be gathered. The planner's `problem.pddl` file contains an 'init' section, which holds the initial state of the storyworld before a plan is generated, made from various predicates which are currently true. To see if a node can be reached from the characters current node the following predicate function is used:

```
(accessible ?to - location ?from - location)
```

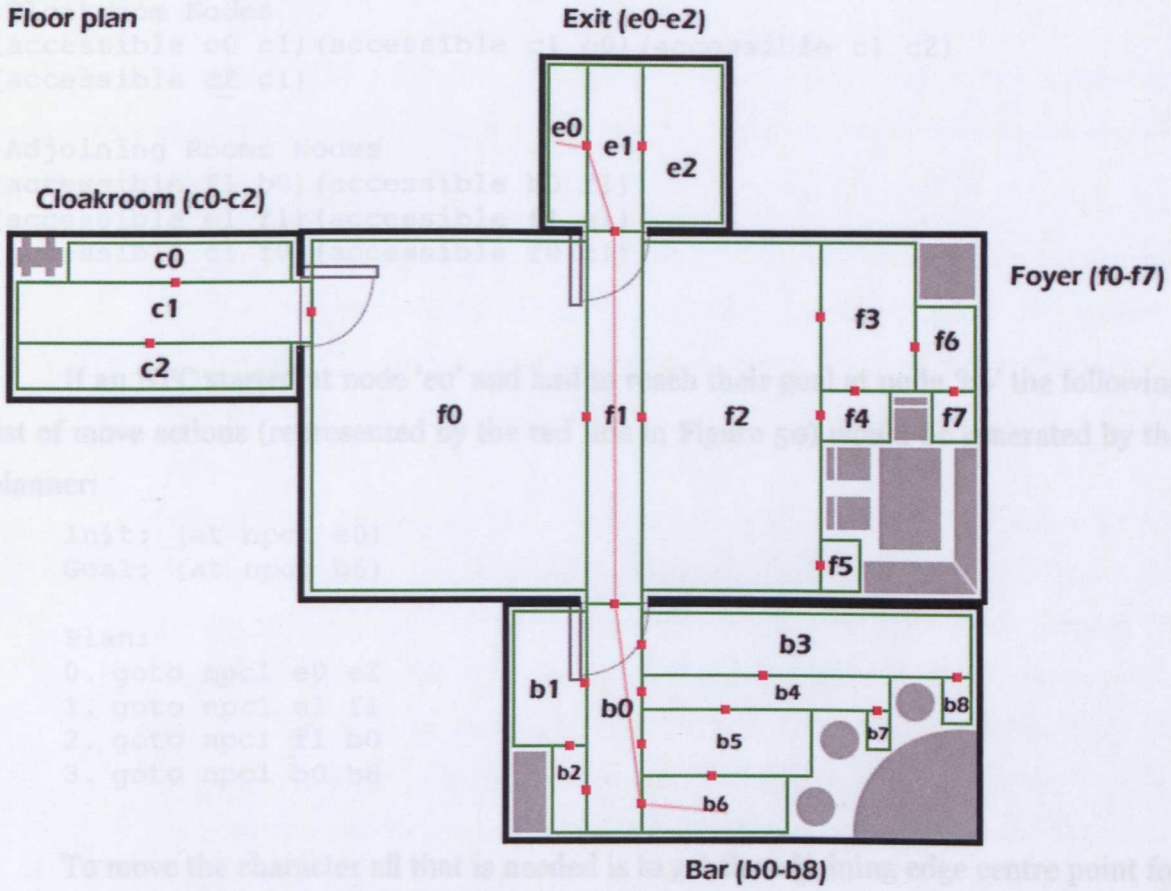


Figure 50: Navmesh Floor plan

Each node is stored as a location in the planner's list of objects. The diagram above contains a total of 23 nodes (b0-b8, f0-f7, e0-e2 and c0-c2), which would be stored as the following init predicates:

```
(:init

;Exit Nodes
(accessible e0 e1)(accessible e1 e0)(accessible e1 e2)
(accessible e2 e1)

;Bar Nodes
(accessible b0 b1)(accessible b0 b2)(accessible b0 b3)
(accessible b0 b4)(accessible b0 b5)(accessible b0 b6)
(accessible b1 b0)(accessible b2 b0)(accessible b3 b0)
(accessible b4 b0)(accessible b5 b0)(accessible b6 b0)
(accessible b1 b2)(accessible b2 b1)(accessible b3 b4)
(accessible b4 b3)(accessible b4 b5)(accessible b5 b4)
(accessible b5 b6)(accessible b6 b5)(accessible b4 b7)
(accessible b7 b4)(accessible b3 b8)(accessible b8 b3)

;Foyer Nodes
(accessible f0 f1)(accessible f1 f0)(accessible f1 f2)
(accessible f2 f1)(accessible f2 f3)(accessible f3 f2)
(accessible f2 f4)(accessible f4 f2)(accessible f2 f5)
(accessible f5 f2)(accessible f3 f4)(accessible f4 f3)
(accessible f3 f6)(accessible f6 f3)(accessible f6 f7)
(accessible f7 f6)
```

```
;Cloakroom Nodes
(accessible c0 c1)(accessible c1 c0)(accessible c1 c2)
(accessible c2 c1)

;Adjoining Rooms Nodes
(accessible f1 b0)(accessible b0 f1)
(accessible e1 f1)(accessible f1 e1)
(accessible c1 f0)(accessible f0 c1)

)
```

If an NPC started at node 'eo' and had to reach their goal at node 'b6' the following list of move actions (represented by the red line in Figure 50) would be generated by the planner:

```
Init: (at npc1 e0)
Goal: (at npc1 b6)

Plan:
0. goto npc1 e0 e1
1. goto npc1 e1 f1
2. goto npc1 f1 b0
3. goto npc1 b0 b6
```

To move the character all that is needed is to get the adjoining edge centre point for the shared edge between each pair of polygon nodes to create the waypoints and restrict movement to the area inside the navmesh. In the diagram the characters path is defined with a straight line, but Catmull-Rom smoothing can be applied for a more natural movement spline path.

The world editor allows the story authors to build a 3d environment or stage/set for their story to take place on. Depending on the type environment required there are specific types of editor modes:

- **Block Mode** (Figure 52) – this allows the user to build up their world from textured blocks by left clicking the face of an existing block to create a new cloned block in the 90 degree compass direction (N, E, S, W) of the selected face. Using the right mouse button the block directly under the cursor can be removed.
- **Vertex Mode** (Figure 52) – this is selected by clicking the centre mouse button and allows the shape of the selected box face to be altered by moving single or multiple vertices using the **transform face tool** (Figure 51: Right), found on the HUD compass. Left clicking will move vertex points towards the camera and right clicking will move vertices away from the camera. A corner point will move only the selected corner vertex; the edge mid-pints markers will move both connecting corner vertices and the centre square will move all four face vertices. The blocks vertices are saved by storing the value of each point, along with its offset value (Figure 51: Left).
- In both block mode and vertex mode the texture coordinate values are also altered to tile the texture to accommodate changes in shape and size.

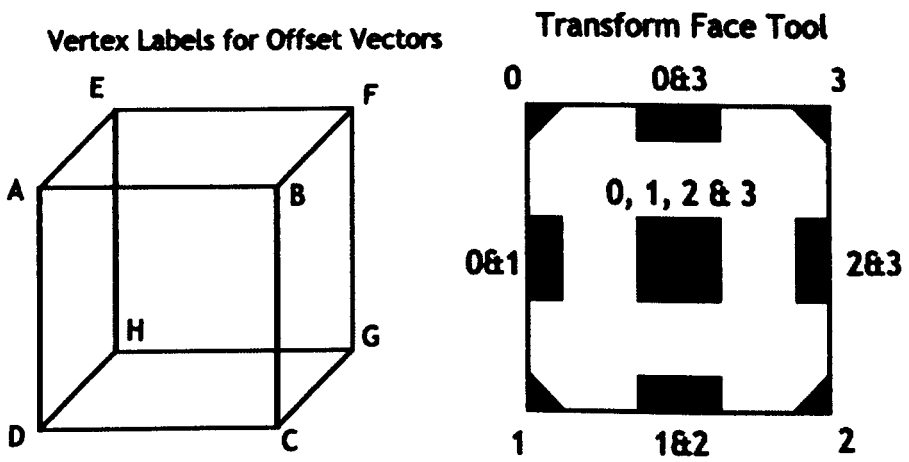


Figure 51: The Vertex Labels for Each Box and Transform Face Tool UI with Vertex Numbers

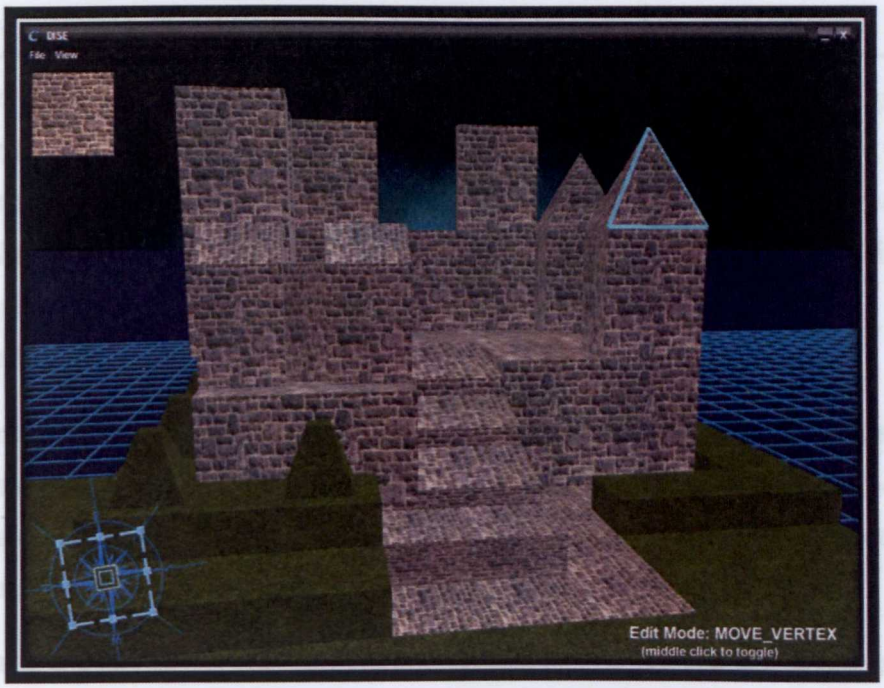


Figure 52: DISE World Editor Vertex Mode

- **Room Mode** (Figure 53) – this editor allows the user to draw squares on the floor plan grid by clicking and dragging with the left mouse button. The editor will then automatically calculate the surrounding walls for the room. The left mouse button adds rooms, or if clicked in an already existing room will extend that room; whilst the right mouse button will subtract the rectangular selection shape from any rooms on the grid. Pressing the space bar allows door frames/archways to be placed on a wall tile with an empty square either side, to create access points and join adjacent rooms and corridors.

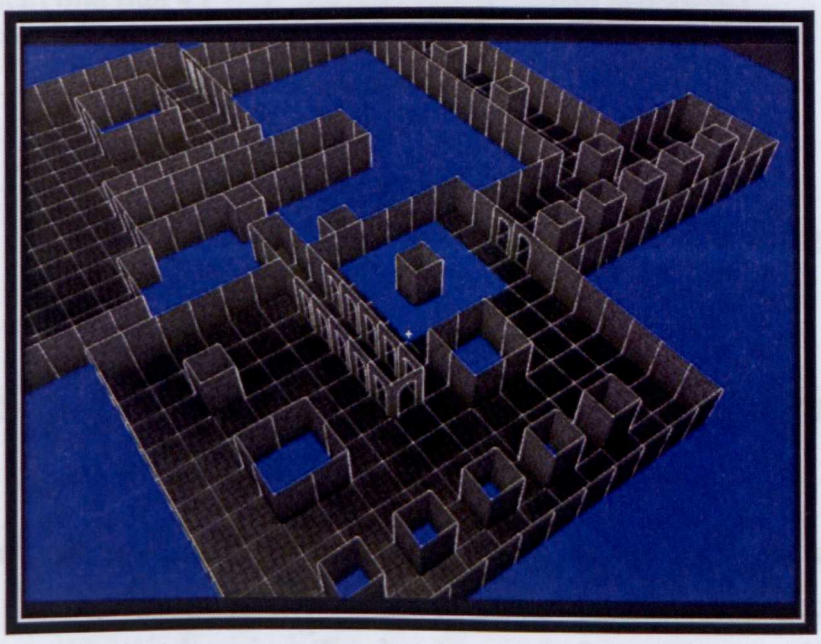


Figure 53: DISE World Editor Room Mode

One of the goals when creating DISE was to create a system that can procedurally generate content for a story location/level, using either random constraints or those set by a user before the generator is started. The final location should then be exportable to a generic file format, which could be loaded in at a later time. This section explains the steps needed to generate a block world environment procedurally with water, hills, trees, buildings and roads.

Environment – To generate a natural environment procedurally we used a variety of techniques, which are broken down into the sections presented below. These sections are the basic parts of a natural world and provide the building methods for land mass/terrain, seas, rivers & streams, and trees/plants.

Terrain & Midpoint displacement fractals – The terrain was created first, as it determines where everything else can be placed and every other part must be built up on top of it.

To create a random terrain we used the midpoint displacement algorithm to create fractal height maps. Four corner points are chosen from the map grid and assigned random colour values between 0 and 1. The square is then subdivided equally into four equal squares and the midpoints of each edge are given a colour value equal to the average value of the two adjacent points (Figure 54). The centre point then gets its colour value from all four corners averaged with the addition of a random offset value, which is scaled by the size of the grid and a roughness variable.

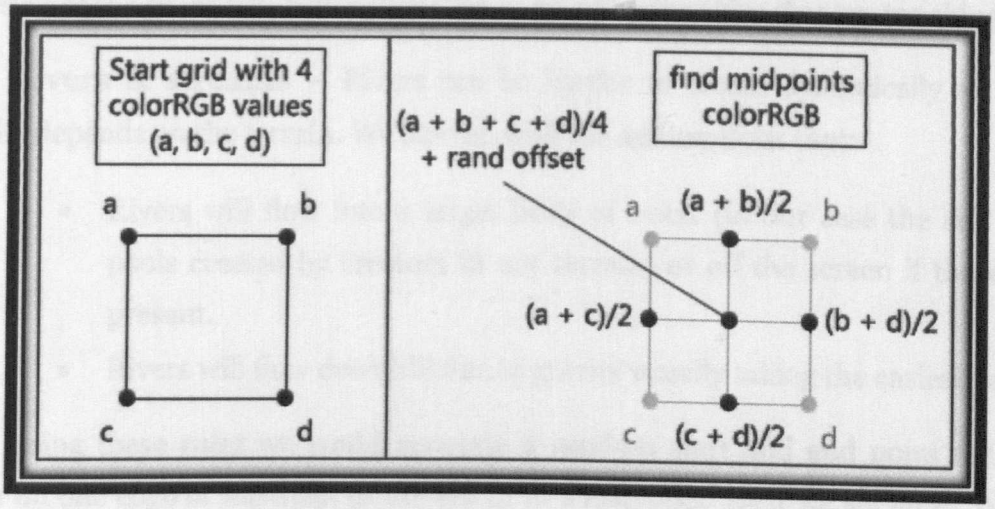


Figure 54: Midpoint Displacement Algorithm

The higher this roughness variable the more dramatic the peaks and troughs will be. This process is repeated to divide recursively until the smallest divisor/unit size is reached, which in this case is 1 pixel. This height map is then output as a black and white image with a grid size of 128x128. These numbers/pixel colours correspond to the terrain

height at point [x, y] (or [x, z] in our 3d game world). The colour values are a percentage of the maximum height of land, with black representing 0% height - the lowest points and white representing 100% height - the high points.

Water – we divided the water creation up into the following three parts:

- The general flat water level created by the sea.
- The edge detection between sea and land creating the shore line/shallows.
- The generation of rivers and streams, that needs to be naturally routed around the map.

Sea and Shoreline – For the sea we created a water level plane; the generated height map for the land mass can determine which parts of the map are above water level and which are submerged. The sea level depends on the range of terrain heights and a water level variable between 0 and 5, which divides the range of terrain heights in to five with zero being no water, five being flooded and the values in between setting the level at 1-4 fifths of the max height.

Once the land and sea were created we added a shoreline of lighter water with an animated tide texture. To implement this we used textured quads placed just above the sea quad. An algorithm scans across each tile in the height map grid that is equal or less than sea level, performing a four-way check to see the surrounding tiles are above sea level. If one direction passes the tile is added to the checked list and not tested further. If the unchecked list has remaining tiles they are checked off one by one until it is empty. Next for each tile in the checked list a new quad was created at the same coordinates with the direction of the shore tiles tide pattern determined by the sides that are touching land.

Rivers & streams – Rivers can be harder to create dynamically as their path heavily depends on the terrain. We started with the assumptions that:

- Rivers will flow into a larger body of water (in our case the sea or in-land pools created by creators in our terrain) or off the screen if there are none present.
- Rivers will flow downhill due to gravity usually taking the easiest path.

Using these rules we could generate a random start and end point for the river, either on one edge of the map, in the sea or in a lake pool. Next we set an A* pathfinding algorithm to compute the route between these two points, using a custom 'cost heuristic', similar to a possible technique for road generation (see the 'Roads' section below for more detail). The heuristic will look for the most cost effective path, with steps taken downhill being cheaper.

Trees & plants – To add more features to our map we added trees and plants. The problem was defining where to place them in a pattern that looks natural. If the trees and plants are scattered using 'n' randomly generated [x, z] positions within our grid the results can look "neither random nor natural" (West, 2008). Due to the way trees and plants grow: from seeds falling; to growing and taking in nutrients, water, CO_2, light and competing with other plants, their scattering pattern can look more complicated. A simple model for this pattern would include each plant having an exclusion zone around it that no other plants can occupy. Plants and trees can be arranged in a grid with an equal distance between them before a small random offset is applied to scatter them. This offset must also be small enough to stop two trees overlapping. More variation can be added later by randomly rotating and scaling the trees (including their exclusion zones), as some trees will be more fully-grown than others.

Roads – Once the land was created it could be populated with an urban city structure. The first main step was to divide up the level into lots (smaller areas that can be used to place buildings and other man made areas such as parks or town squares) by forming a road network. There are two main structural layouts for the roads, which were dependent on the area being in an urban or rural setting. For a big city the gird plan road structure is most fitting, but in rural areas the streets need to be non-uniform (smaller, more winding and with further separation). As the land and natural features are determined first, the height map from terrain generation could be used to control where the roads can and cannot be built. Features like water and hill gradients can be factored in to exclude roads or define points for bridges and tunnels.

There are two useful techniques we could use to generate the roads: L-systems and A* Pathfinding. These could be used individually to give different results or be used together in passes to create roads more intelligently.

L-Systems – An L-system or Lindenmayer system is a parallel rewriting system using string rewrites, where each symbol is interoperated as a rule step. They can be used to generate self-similar fractals. An L-system contains: variables, constants, a start string (*start*, *axiom* or *initiator*) and rules. The original L-system was used to model the growth of algae and can be seen in Figure 55 below.

This L-system contains:

Variables: A B

Constants: none

Start: A

Rules: (A → AB) (B → A)

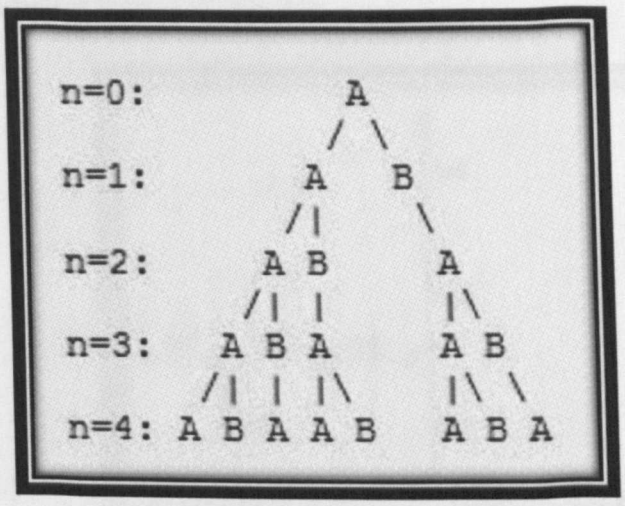


Figure 55: Original L-System for modelling the growth of Algae

We used this system to create our roads starting with a small line drawn in the initial forward direction. As the steps for 'n' increased more lines were added and for each one a random number check was done with a small chance of returning true. If the test returns true up to three branches will be created in the north, east and west directions from the original point at around 90 degrees to create a block pattern (this number could be changed, or have a random offset to give different road patterns).

To create a rural non-linear road system the distance for each line drawn every 'n' could be shortened and the maximum number of branches reduced from 3 to 2. Also as mentioned previously the chance of the angle changing could be increased and the angle value varied to create a more curved or 'dog-leg' shaped road.

To make the road creation more dynamic we overlay the height map from the terrain whilst building the road (Lintfordpickle, 2009). Areas that have water could be impassable or require a bridge and the gradient can be checked to stop roads having too steep an incline and force them to spiral uphill gradually. We could also add a population density map, which can control where main roads are build and the density of the roads. These rules can all be added to the drawing process 'A' mentioned above and alter the return true statement for adding new branches.

Our L-system is structured in the following way (Figure 56):

Variables: X A

Constants: + −

Start: X

Rules: (X → [[-AX] +AX] + AX)

Angle: 90°

X = road growth control variable

'[' = Push

']' = Pop

'-' = Turn Left Angle°

'+' = Turn Right Angle°

'A' = if random check true draw line of distance 'D' in forward direction.

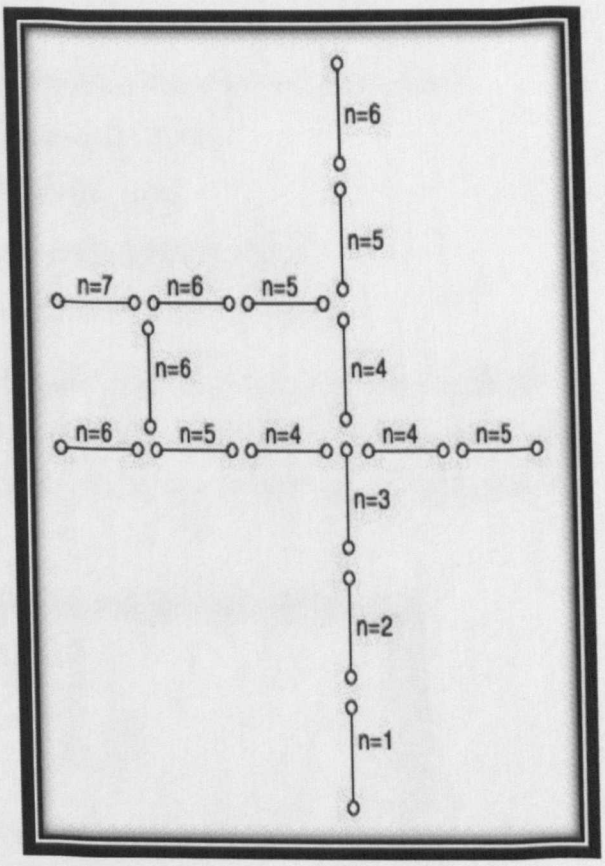


Figure 56: L-system for grid block system roads

A* algorithm and heuristic pathfinding – Pathfinding is used to navigate from one point to another whilst avoiding any obstacles in the way by finding a suitable route around them. A popular algorithm for this is A*, which can run in real time on a reasonable sized map. As our roads are generated in an editor a slight delay is not as big an issue as it would be in other cases such as an enemy AI in a shooter game. We can scatter points around the map then connect them using pathfinding to create the roads along suitable routes (Glass) (Lester, 2005).

A heuristic developed by multiple authors for the game 'Open Transport Tycoon' devised a logical cost heuristic for a road-building pathfinder, which also includes provisions for turns, slopes, bridges and tunnels (OpenTTD Contribs., 2011).

Buildings – When the roads are placed the shape of the lots between them can determine where the buildings can go.

Shape grammars – Shape grammars are used to provide a computational approach to the generation of designs. In architecture, building structures have many repeating shapes and patterns that can be described using a rule.

Shape grammars consist of:

- Shapes – the structures building blocks.
- Spatial Relations – how these shapes are positioned relative to one another.
- Rules – to describe how and where new shapes are added.
- Rule Labels – give a set orientation marker to the rule.
- Transformations – translation, rotation and scale of the shapes.
- Derivation – repeat the rule for n steps to create multiple designs.

Figure 57 shows a labelled rule for a rectangle. For each step a new rectangle is added with a clockwise rotation of 90° and aligned to the original shapes narrow edge (labelled with a dot). The pattern repeats for 3 steps, forming a closed square shape with a gap in the centre.

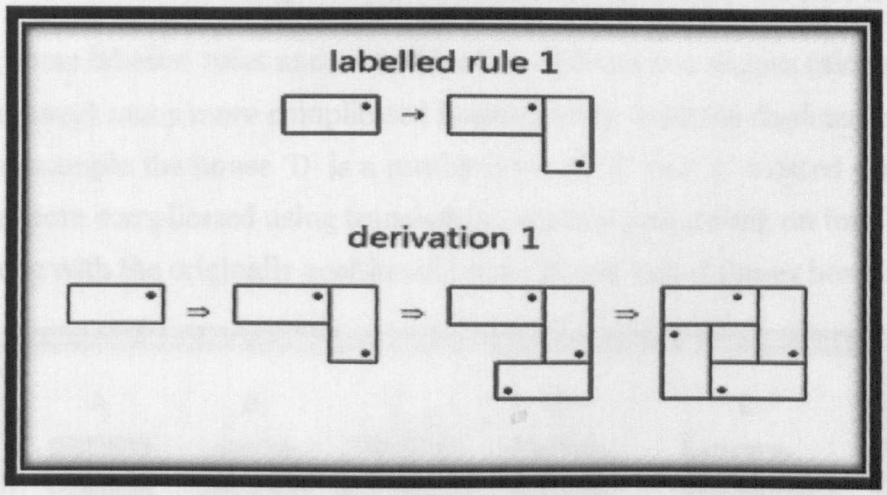


Figure 57 Labelled Rule Example 1 (Knight, 2002)

Figure 58 shows that with exactly the same shapes that changing the label dot, thus changing the rule, can produce a different structure. This time the new rectangle is rotated 90° in an anticlockwise direction and aligned with the short marked edge.

Figure 58 Labelled Rule Example 2 (Knight, 2002)

Using these examples with randomly sized boxes can produce some interesting tower block shapes for the city buildings. The number of steps n can be also set randomly, so just by using the two rules above 8 different structures can be made and when combined with the randomly sized boxes this creates hundreds of different buildings.

To create cylindrical office buildings we consider the technique used in (Young, 2009). A circle is generated in 10° slices, with a random chance of skipping ahead 90°, creating a bigger slice and a flat edge. This process continues with up to three 90° skips, until the total rotation is a full 360° circle. The circle can then be extruded upwards with a random height value and capped to create a cylindrical office building.

We experimented with shape grammars to also create smaller houses and other buildings such as a church. Figure 59 below shows two basic house shapes (A & B) created in 3DS Max and loaded into our game engine as a shared mesh which can be cloned.

Using some labelled rules and a combination of these two shapes (along with a thin box for a chimney) many more complicated houses can be built (as depicted in C, D, E, F, G & H). For example the house 'D' is a combination of 'B' and 'A' rotated 90° clockwise. House 'H' is more complicated using translation, rotation and scaling on four instances of house 'A' along with the originally positioned house 'B' and two chimney boxes.

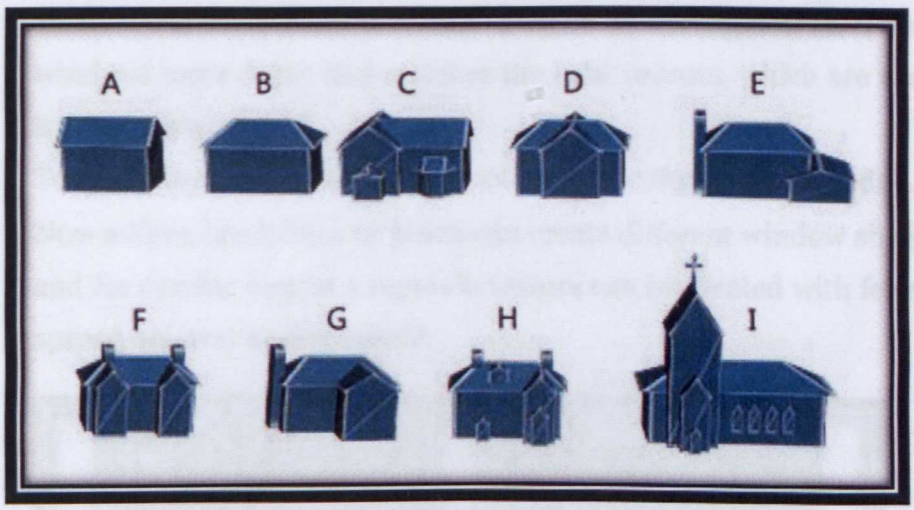


Figure 59: Example House Configurations

Procedural Textures – The textures used in the scene are generated in two main ways for the land, trees, water and buildings:

- Existing textures produced beforehand by an artist in the PNG format are dynamically chosen and scaled according to the object and a set of rules.
- The texture is created procedurally using AWT image graphics and saved as a texture state, which then can be treated as the predesigned textures above.

To create the textures for our buildings we used the latter option to create a series of

windows in different sizes, colours and designs. To make the process easier to start off with, we used a procedure similar to the one described in (Young, 2009), where buildings are dark to mask the missing detail and defined by lit windows, which gives a night time city effect. A 512x512 texture is reserved in memory and each pixel is coloured in several stages:

- First determine the number of windows across and down and what their size is when equally dividing up the texture.
- Set a base colour for the window that is dark (unlit), white or orange (lit). These colours also have a random offset to give a small variation in tone.
- There are two settings for the distribution of the lit windows to simulate either residential tower blocks or offices. The residential buildings have a random scattering; the lit and unlit windows are selected using the math library's next random integer function, with the probability set to roughly 1:3 lit to unlit and an equal chance for white or orange to be chosen for the lit windows. The offices are lit in random blocks consisting of up to a whole floor of windows as companies usually occupy a whole floor or several units and some are open later than others.
- Next for each pixel row the selected base colour is modified by subtracting a small value each time to create a light to dark gradient. This gives the windows more depth and matches the light sources, which are usually at the top of a room.
- To create more variation a per-pixel random colour noise is added.
- Now adding black lines or pixels can create different window shapes/designs and for smaller houses a separate texture can be created with fewer windows spread out over a wider space.

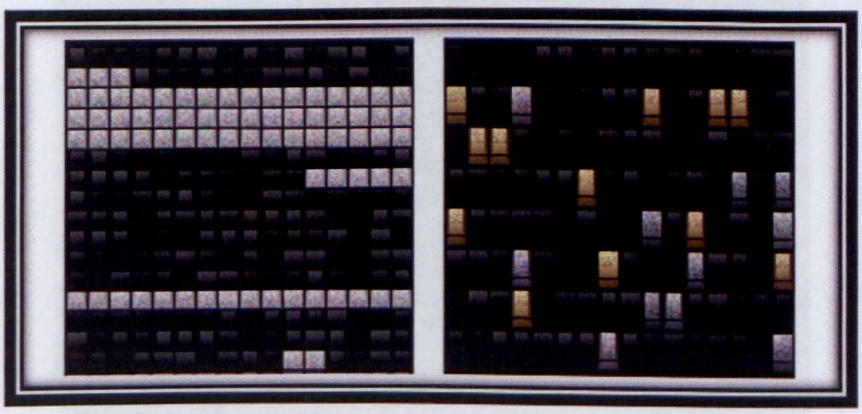


Figure 60: Procedural Window Textures

To create more variety we considered an assortment of window shapes and sizes, especially for the smaller houses textures. Houses have less windows and light, with a

more varied array of designs including the bay styled windows.

Implementation – we implemented the user interface using swing, with a similar look and feel to the Story Editor. Some controls are provided to the user to constrain elements of the level generation, including the option to turn off some generated features in a level or completely randomise the generation for a two-click level design.

Figure 61: Swing User Interface Layouts

Some example settings that would be useful are:

- Terrain Height – flat/large peaks.
- Terrain Roughness – flat/many peaks & troughs.
- Water Height – no water/some water/flood.
- Tree Density – no trees/ many trees
- Texture theme – Grass, Desert, Snowy
- Rebuild Level – Rebuild geometry using the above settings.
- Show Tweening Animations – on/off.
- Buildings – on/off
- Population Density - City/Rural
- Roads, Bridges, Tunnels – on/off toggle each

Figure 62: Procedural Buildings

Optimising geometry – To optimise the creation of our geometry we used shared meshes. There are many objects in the scene that are copies of each other and shared mesh allow us to share that data between multiple nodes. "A provided TriMesh is used as the model for this node. This allows the user to place multiple copies of the same object throughout the scene without having to duplicate data" (Powell). Each copy node has unique translations, rotations, scales and render-states applied to them. Another useful feature to include is object pooling. We initialised a store of objects in a pool when the game starts for objects like new buildings, trees and textures; so that when one is created it can be quickly taken from this pool.

Figure 63: Procedural Terrain Generator

5.5 STORY MANAGER

In our framework the Story Manager contains all the information that will progress the narrative according to the story authors design. All the effects of the player's and NPCs actions are sent back to the Story Manager via the World Fact Database, which it monitors for changes and then checks against its internal predicate logic waiting for a trigger to manipulate the story data. The Story Manager class is also responsible for creating/instancing the objects and assets needed for the current scene and propagating these changes to the relevant classes and data stores.

5.5.1 SCENES

To create a story, we have taken inspiration from the more mature media of literature, film and theatre and use their tried and tested structure of 'sequences' and 'scenes'. A film's scene is described as "a shot (or series of shots) that together comprise a single, complete and unified dramatic event, action, unit, or element of film narration, or block (segment) of storytelling within a film, much like a scene in a play; the end of a scene is often indicated by a change in time, action and/or location" (Dirks, 2010), with a sequence being a number of consecutive scenes that have an overarching plot. In movies if the scene has no purpose or justification it is cut. Using these scenes and giving the story author a Scene Editor to create them and their hierarchical structure, DISE give a higher level of flexibility than enforcing a particular narrative theory. For example scenes can be used to recreate the structured acts mentioned in section 3.2.2 FIVE-ACT MODEL and 3.2.3 THREE-ACT PARADIGM or even in a more modern way like the non-linear stories such as Pulp Fiction mentioned in 3.2.4.

In fiction writing the job of the scene is the same as our scene concept for Interactive Storytelling to achieve as many of the following goals as possible (Bickham, 1993):

- **Advance the story** – a scene needs to move the story forward; it can also introduce problems or further them.
- **Create conflict** – this could be between characters or against time, nature, environment, etc.
- **Introduce a character** – a new character can be introduced to the story.
- **Develop a character** – reveal more of a character's personality and intentions.
- **Create suspense** – building up some minor events with large pauses before something big happens.
- **Giving out information** – pass some information to the player via text, speech or visuals.

- **Creating an atmosphere** – changing the current environmental properties and time can set a tone for the story (e.g. a horror could have the creaky mansion at night time in a thunder storm).
- **Develop the story's theme** – the story should have a main theme which can be further revealed to the player in a scene.

As well as definite purpose a scene also needs a start and end, a point of view, a setting (location), a following scene and a length of time to play out over.

Although in the aforementioned media these scenes are linked linearly we can use them graphically to more easily create a multithreaded story made up of bite-sized chunks. We want our stories to be dynamic and allow the player to have an impact on their decisions and eventually the outcome of the story. In DraMachina, Donikian and Portugal logically define their narrative scene objects as "the combination of current setting/actors on stage/Dramatic Action currently going on/and present state of the Dramatic Units map. Entrance of a character, change of Dramatic Action... will automatically change the Scene Object and thus its parameters such as Mood/Ambience, Level of Formality, Rhythm of Action, Actions Allowed, etc." (Donikian & Portugal, 2004).

This automatically changing scene model can be replicated by building triggers into scenes that call other scenes, creating our dynamic sequences.

Each scene contains the following information to give it a purpose and to progress the story further:

- List of verbs (actions) available to the player in that scene.
- List of StoryMods (character intro/individual events/changes to the story).
- List of Trigger Connectors (the events that when triggered will advance to the next scene e.g. "a change in time, action and/or location" (Dirks, 2010) as mentioned above).

By choosing the selection of verbs/actions a player can execute in a given scene and running several StoryMods an author can craft a story that indirectly affects the player giving them more freedom to explore and make their own choices whilst still under some of the author's constraints (for example in a fairy-tale the verb 'slaying' (of the dragon) can be withheld until the hero has reached the 'quest attained' scene in the story. By creating scenes in the editor and linking them into sequences with triggers for each one the story author is creating a high level finite state machine (FSM) that modifies the story differently depending on the player's actions (Figure 64).

FSMs are a well-formed programming technique used already in many games today, so they are relatively easy to represent in code and also to save out in a file from the editor

to be loaded in as a story file at a later time. The novelty of this system compared to other FSM based games is that the state machine is only used to indirectly guide the story towards certain events, for example: an author could create a scene that is only triggered when the king is angry, but what actually makes the king angry is not pre-determined. This would allow a number of events to change the king's emotional state and the player could even avoid the scene completely if the king's happy mood is maintained.

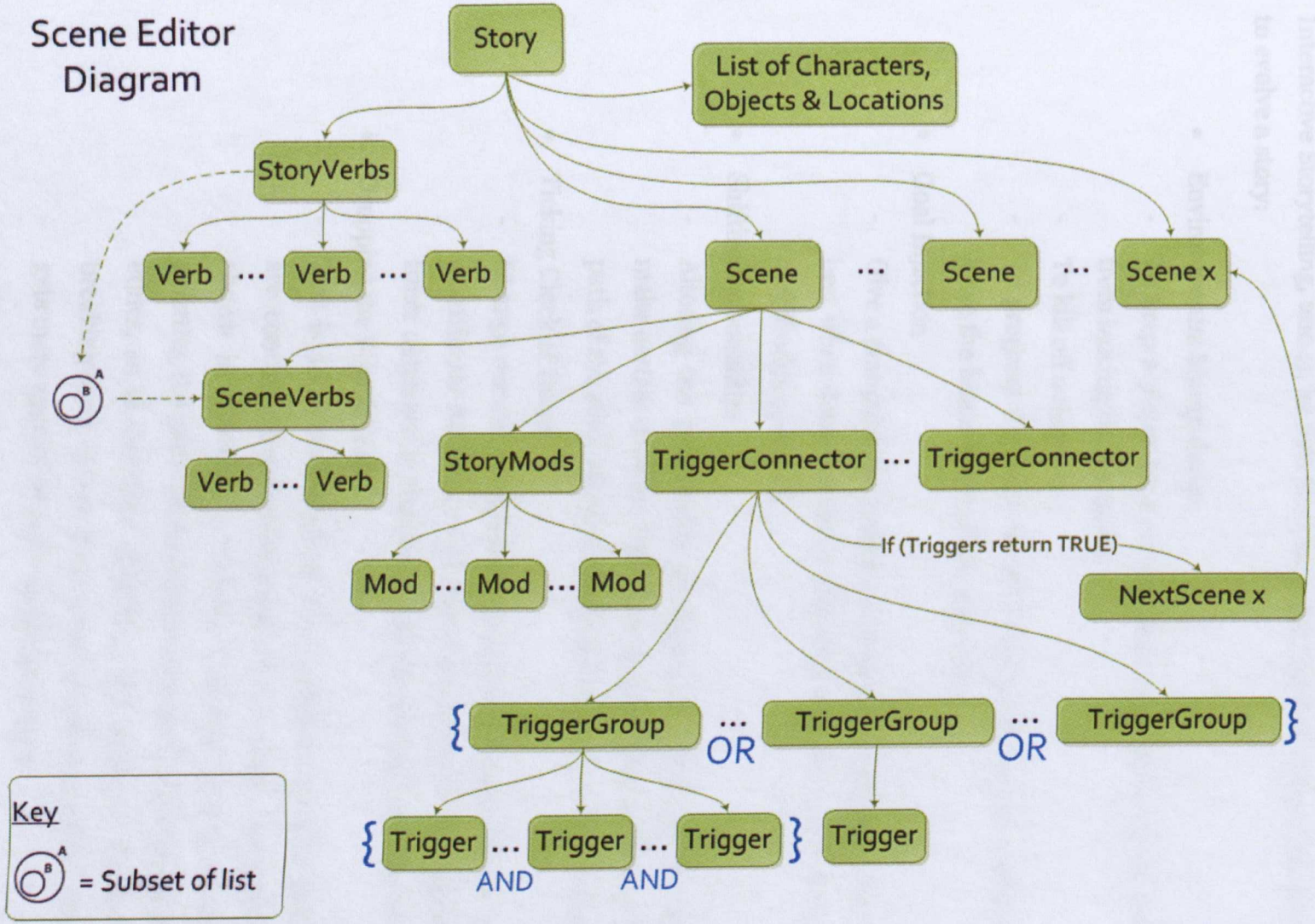


Figure 64: Scene Editor Diagram

5.5.2 STORY MODS

When a scene is triggered, it will execute all of its StoryMods in order. A StoryMod modifies the story in some way to progress the narrative. We want to create a base library of StoryMods with the option for programmers to extend them if necessary.

In Crawford's book on Interactive Storytelling (Crawford, Chris Crawford on Interactive Storytelling, 2004), he lists the following techniques that can be implemented to evolve a story:

- Environmental Manipulation
  - To keep a player on a certain path, e.g. a cave-in can prevent players from leaving the dungeon.
  - To kill off a character.
  - To progress the story or alert the player, e.g. at sunrise they should leave the house to set off on a journey.
- Goal Injection
  - Give a computer character a new goal to change the story. This works best when done in context adhering to a characters traits, personality and Modus operandi.
- Shifting Personalities
  - Altering the personality variables of NPCs can influence them to make certain choices. This is an elegant way of discreetly altering the path of the story without breaking the player's immersion.
- Ticking Clock of Doom
  - Change certain variables gradually over a period time, to converge to a particular state or event. This should also be done discreetly with no timer displayed to the player, unless needed for a particular story.
- Dropping the Fourth Wall
  - This is a last resort, where the players are alerted that their actions are contradicting development of the story. An example of this is shown in Nintendo's 'Animal Crossing' game, where deliberately resetting the game circumvents the intent of gameplay mechanics and ethics, so is therefore cheating. The player is alerted of this 'rule breaking' by a lecture from a mole character called "Mr Resetti" that gets more aggressive with each appearance.

We adapted this guide list comparing it to the goals/purpose of a movie scene listed in the last section (5.5.1) and created the following StoryMods to cover each technique:

- **Advance the story**
  - Progress time forwards, e.g. `(increase (time-days) 1).`
  - Display Message Text t, e.g. `"you wake up a day later and find out that your magic cloak has gone".`
  - Add/remove props, e.g. `remove: cloak.`
  - Add/remove facts, e.g. `add: (islit b0).`
  - Add/remove characters, e.g. `remove: npc1.`
  - Add/remove character goals e.g. `remove: (have-read npc1 message).`
  - Change character personality values, e.g. `(decrease (AngerFear npc1) 1).`
  - Display a Charisma scene animation. E.g. animated character dialogue explaining why the quest is important.
- **Create conflict**
  - Add/remove props, e.g. add things that hinder the player or remove helpful items that need to be reclaimed.
  - Change characters personality & goals to create conflict (see Develop a character below).
- **Introduce a character**
  - Character enter/leave story.
- **Develop a character**
  - Change personality variables, e.g. `(decrease (AngerFear ?npc1) 1)` or `(perAttractive ?npc1 - character ?npc2 - character).`
  - Give character a new goal, e.g. `(have-read npc1 message).`
  - Display a Charisma scene animation.
- **Create suspense**
  - Add/remove facts, e.g. add `(islit b0)` would turn on the light in bar area zero.
  - Add/remove props.
  - Display Message Text t.
- **Giving out information**
  - Display Message Text t.
  - Display a Charisma scene animation.
- **Creating an atmosphere**

- Add/remove facts, e.g. add `(islit b0)` would turn on the light in bar area zero.
- Add/remove props.
- Display Message Text t.

- **Develop the story's theme**
  - See advance the story above.

This base list allows some freedom to create a variety of stories for example: The long lost heir to the throne enters the story and has the goal to reclaim the kingdom. While the current evil ruler gains the new goal to kill the heir and has an increased value for anger.

5.5.3 STORY TRIGGERS

In the last section we talked about StoryMods but the StoryMods in a scene will not be executed until their encapsulating scene is called, this is where Story Triggers come into play. Story triggers are created in the editor by joining two scenes together with a trigger connector (shown in Figure 65 below with the alarm clock icon). Each scene can have multiple connectors going in and out, but each connector can only have two scenes attached creating a "TO" and "FROM" link. Selecting a trigger icon brings up the trigger manager menu.

This menu shows all the triggers that need to equate to true for the "TO" scene to be activated. Individual triggers can be stacked up and resolved using propositional logic and 'disjunctive normal form' (Russell & Norvig, 2003) to create more complex trigger sentences. Triggers stacked in the "WHEN" box are automatically put in a trigger group "AND" block. To create an "OR" statement new triggers should be added to the "OR WHEN" box, in the same way as using production rules.

This allows expansion of statement creation for each trigger connector. The triggers have a shared java interface containing their Boolean value to speed up test calculations. Many of these tests can be run in quick succession, but the greater the number of scenes and triggers branching from one scene the greater the impact on performance of the system. In the future we need to benchmark the scalability of this performance and may have to set limitation caps or cost values to the number of links from any one node.

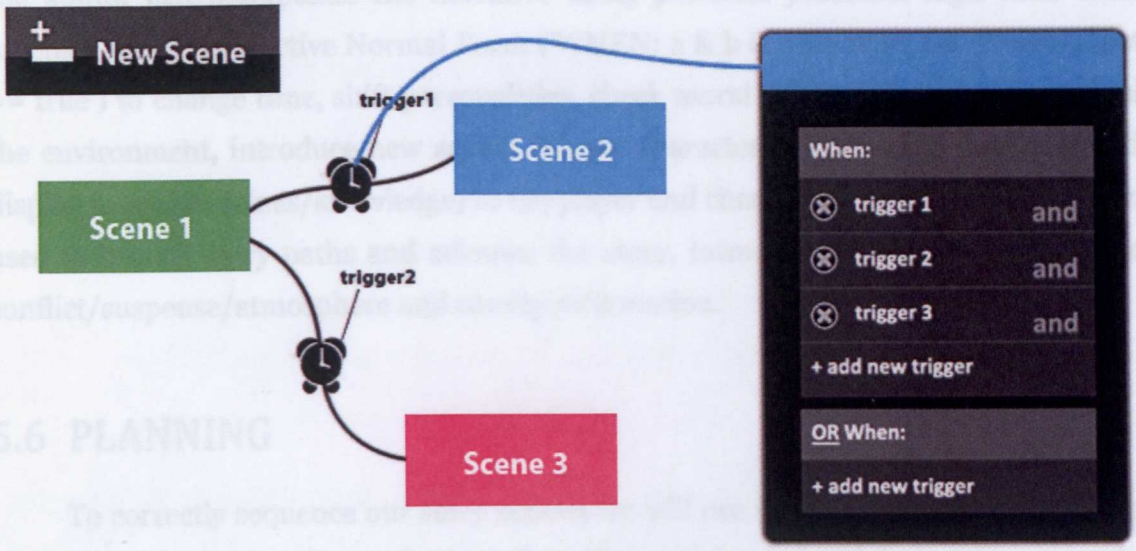


Figure 65: Story Editor and Scene Trigger Manager

The current list of story triggers contains the following types, but can also be expanded if necessary:

- Time (exactly t, after t, before t)

```
If time == t
If time > t
If time < t
```

- Personality

```
If (Attribute Character) == value
If (Attribute Character) < value
If (Attribute Character) > value
e.g. (=(AngerFear bob)0)
```

- Event

```
If (Verb Character Noun…) == (v, c, n…)
e.g. (scare player1 bob)
```

- Character and/or Object at/not at Location

```
If (at Character/Object Location) == t/f
e.g. (at bob bar)
```

- Object State (fact does/doesn't exist in database)

```
If predicate(param 1, param 2, param n) == t/f
e.g. (wearing player1 cloak)
```

5.5.4 *STORY MANAGER CONCLUSION*

The Story Manager actively checks the state of the story world and progress through the author's scene structure using Story Triggers to link consecutive scenes. Its novel design allows the author to use multiple narrative theories (for example three-act and five-act models) or even their own scene structure to progress their story. Using the StoryMods

the author can manipulate the narrative using powerful predicate logic facts chained together using Disjunctive Normal Form ('WHEN: a & b & c == true; OR WHEN: [not] d == true') to change time, shift personalities, check moral values, inject goals, manipulate the environment, introduce new agents, trigger character dialogue and facial animation, display messages (clues/knowledge) to the player and change any state. These tools can be used to branch story paths and advance the story, introduce/develop characters, create conflict/suspense/atmosphere and convey information.

5.6 PLANNING

To correctly sequence our story actions we will use a branch of artificial intelligence called automated planning and scheduling. Planning is used widely in computer science for problem solving, timetabling and controlling intelligent agents. The player knows to open a locked door they need to pick up a key first, but a complex system is needed to get a non-player character to procedurally carry out this sequence of instructions thus reaching a goal state.

5.6.1 PDDL

Planning languages have recently been standardised in 1998 to a single description language called PDDL (Planning Domain Definition Language) (Haslum, 2003). PDDL is made up of smaller optional construct modules, but not all planners implement the whole set. The main construct of PDDL is called STRIPS (Stanford Research Institute Problem Solver) after the automated planner that used the language as its input to describe classic planning tasks. To create a planning task in PDDL the following components are required:

- Objects
- Predicates
- Initial State
- Goal Specification
- Actions/Operators

These components are distributed within two files: the **domain** file, which contains the predicates, and actions and the **problem** file containing the objects, initial state and goal (Figure 66). Also shown in Figure 66 are extra domain details such as construct requirements, type hierarchies, and number based functions.

Predicates have shared traits in human language and computer language and are an expression that is true of something, expressing a relationship, or property of an argument in a clause. Predicates are the second main part of a sentence, with the other being the subject that is modified. In PDDL predicates can have multiple subject parameters along

with their corresponding types.

Actions are usually made up of three parts, similar to the example shown in section 5.4.7:

- **Parameters** – Usually the 'Name' and 'Type' of an object, character, or place that are used by the action
- **Preconditions** – Requirements that need to be met to perform the action.
- **Effects** – Post conditions & states after the action is carried out.

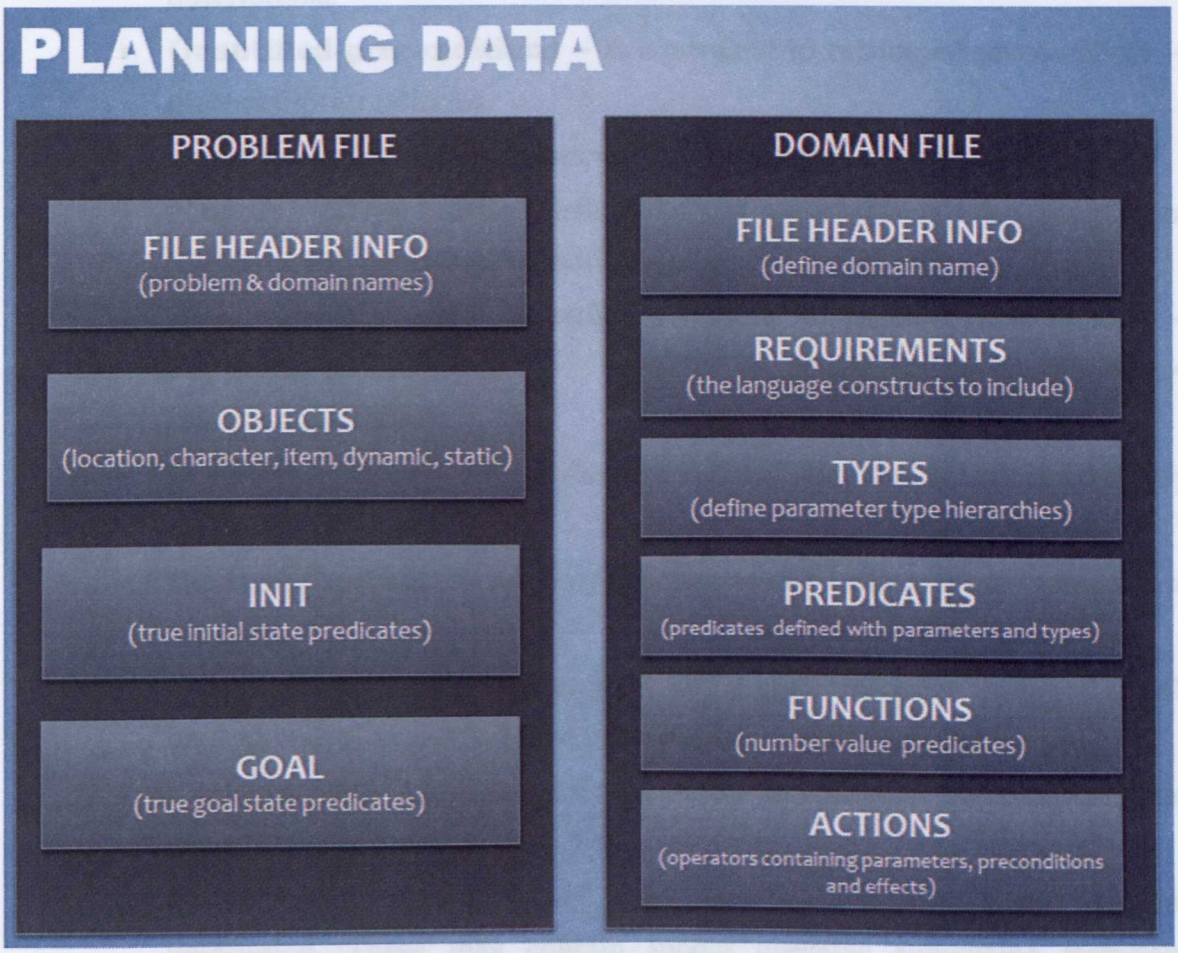


Figure 66: Planning Data Requirements

There are several versions of PDDL, with PDDL 3.1 being the latest version used in the International Planning Competition IPC 2011 (Jimenez, 2010). This version includes the same features of the previous versions but has more optional modules to provide extra functionality to the language and are included using the "`:requirements`" definition in the planning domain file. The functions of these

Features of PDDL 3.1:

- **Strips** – Basic STRIPS-style.

- **Typing** – Allows type names in declaration of variables.
- **Negative-preconditions** – Allows not in goal and preconditions descriptions.
- **Disjunctive-preconditions** – Allows or in goal and preconditions descriptions.
- **Equality** – Supports '=' as built-in predicate.
- **Existential-preconditions** – Allows exists in goal and preconditions descriptions.
- **Universal-preconditions** – Allows for-all in goal and preconditions descriptions.
- **Quantified-preconditions** – Is equivalent to existential-preconditions + universal-preconditions.
- **Conditional-effects** – Allows when clause in action effects.
- **Fluents** - Allows function definitions and use of effects using assignment operators and numeric preconditions.
- **ADL** - Is equivalent to strips + typing + negative-preconditions + disjunctive-preconditions + equality + quantified-preconditions + conditional-effects.
- **Durative-actions** – Allows durative actions. Note that this does not imply 'fluents'.
- **Derived-predicate** – Allows predicates whose truth value is defined by a formula.
- **Time-initial-literals** – Allows the initial state to specify literals that will become true at a specified time point implies durative-actions.
- **Preferences** – Allows use of preferences in actionCtx preconditions and goals.
- **Constraints** – Allows use of constraints fields in domain and problem description. These may contain modal operator supporting trajectory constraints.

To run DISE stories using different planners the planner interface class was created as a flexible abstraction layer, but DISE is still constrained to using the PDDL structure for the problem and domain definitions as the logical structure of PDDL's goals and actions is incorporated into the core action class and also the general concept of DISE. A new planner can be added by writing a new planner link class or changing the existing one to fit the function call and arguments of the new planner. The current planner is called via the command line using the Java function:

- `Runtime.getRuntime().exec`

- This function takes the following arguments:
- Planner exe name (including folder name inside 'planners/resources' package), operator (domain) PDDL file, facts () PDDL file.
- `Planner_folder_name/planner_exe -o operator_file.pddl -f facts_file.pddl`

In this section we defend the importance of planners and explain their usefulness in Interactive Storytelling systems. First we look at why they are used and what benefits they provide; next we evaluate the language constructs of PDDL and filter out which ones are necessary. The next section briefly covers search optimisation needed for planning to run in real-time and how durative actions can allow concurrent planning in future implementations of DISE. Lastly we review some benchmarks done on existing planners by Barros and Musse, showing their feature set and speed.

Why Use Planning? – Orkin created a STRIPS style planning system for the AI enemies in the game F.E.A.R. (Orkin, 2006). This technique is a good fit in many ways:

- New goals and actions can be easily added to extend the support for different stories and genres "with a planning system, we can just toss in goals and actions. We never have to manually specify the transitions between these behaviours. The A.I. figures out the dependencies themselves at run-time based on the goal state and the preconditions and effects of actions." (Orkin, 2006)
- The STRIPS planning syntax fits well with the creation of action sentences, using production rules, as each action is stored in a format describing its parameters, preconditions and effects in key: value pairs.
- When an object is chosen to interact with we can search in each action's parameter definition for verbs that support objects of the same type.
- Using the effects the game state can be updated after an action is performed.
- Preconditions can be used to remove actions that are not important in the current context.

Constructs – There are many planners available that use this PDDL input and read the planning problem and domain from two separate files. The only problem is that the full specification of PDDL 3.1 is not supported by all planning systems. From the list in section 5.6.1 Barros & Musse (Barros & Musse, 2007) consider the following five extra language-constructs valuable to Interactive Storytelling applications:

- Type hierarchies – each object and parameter is given a type classification to avoid errors.
- Equality – uses the '=' sign to mean equals, which is useful for conditional effects value tests e.g. `(when (= ?x cloak) (islit ?z))`
- Negative preconditions – allows the 'not' command to be used with conditionals and equality, for example to describe actions for when a

character does not know a fact.

- Conditional effects – allows the use of conditional 'when x > do y' statements.
- Existential preconditions – "Using an existential quantifier in the precondition, we are able to state that the examiner character can only perform this action if he is alone in the examined place".

This means that when attaching a planner to DISE it must be checked to contain the correct constructs, or will not be complete enough to process detailed stories.

Search Optimisation – Another factor in planning for computer games is search optimisation. If new plans are to be generated in real-time alongside processor intensive game logic and rendering, they have to be fast to execute and highly optimised.

There are many different search techniques which are widely used today in many areas of computer science, including:

- Forward chaining.
- Backward chaining.
- State-space search, with conditional relationships or heuristics (see graphplan).
- Propositional satisfiability (satplan).
- If the assumption of determinism is dropped and a probabilistic model of uncertainty is adopted, then this leads to the problem of policy generation for a Markov decision process (MDP) or (in the general case) partially observable Markov decision process (POMDP).

Durative Actions – Durative actions (Fox & Long, 2003) give a specific duration to an action and categorise the time conditions are tested using three time groups: at the start of the action, at the end of the action and over all the whole action. The actions effect is also labelled as 'at start' (for immediate effects) or 'at end' (for delayed effects). This extra set of information allows concurrent planning (the use of resources by multiple actions). The example below shows the durative action for burning a match whilst simultaneously picking up a coin.

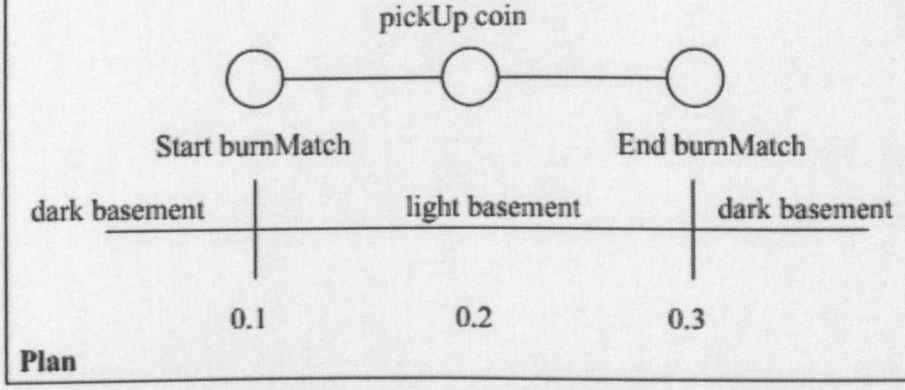
```
(:durative-action burn-match
:parameters (?m - match)(?l - location)
:duration (and
(< ?duration 5)(> ?duration 0)
)
:condition (and
(at start (have ?m))
(at start (at ?l))
)
:effect (and
            (when
```

```
                    (at start (dark ?l))
                    (and
                    (at start (not (dark ?l)))
                    (at start (light ?l))
                    )
               )
               (at start (not (have ?m)))
               (at start (burning ?m))
               (at end (not (burning ?m)))

               (when
                    (at start (dark ?l))
                    (and
                    (at end (not (light ?l)))
                    (at end (dark ?l))
                    )
               )
     )
)
)

(:action pickUp
:parameters (?l - location ?o - object)
:precondition (and
(at ?l)
(onFloor ?o ?l)
(light ?l)
)
:effect (and
(not (onFloor ?o ?l))
(have ?o)
)
)
```

Initial state: (onFloor coin) (have aMatch) (at basement) (dark basement)
Goal: (have coin)

Problem

Plan: 0.1 (burnMatch aMatch basement) [0.2]
0.2 (pickUp basement coin)



Plan

Figure 67: Concurrent planning using durative actions (Fox & Long, 2003)

Benchmarking Speed and Feature Sets – Barros & Musse (Barros & Musse, 2007) benchmarked different planning algorithms testing their speed and feature sets with Interactive Storytelling applications in mind. From this table we decided to test DISE first with the Metric-FF planner, as it contains the necessary language constructs and has a desirable speed. The nature of DISE allows other planners that use PDDL to be switched in if the converter class is written to translate the appropriate values to and from the planner, so these results could be cross referenced with our benchmark.

Table I. Summary of Algorithms

Algorithm	TH	EO	NP	CE	EP	Opt	POP	AC	NV	Time
FF	✓	✓	✓	✓	✓	×	×	×	×	0.015 s
Graphplan	×	×	×	×	×	✓	✓	×	×	0.138 s
HSP	✓	✓	×	×	×	×	×	×	×	0.031 s
HSP*	✓	✓	✓	×	×	✓	✓	✓	✓	3.641 – 7.25 s (1)
IPP	✓	✓	✓	✓	✓	(2)	✓	×	×	0.106 / 0.032 s (3)
LPG-*TD*	✓	✓	✓	×	✓	×	✓	✓	✓	0.680 / 0.169 s (4)
Marvin	✓	✓	✓	✓	✓	×	✓	×	×	0.017 s
Metric-FF	✓	✓	✓	✓	✓	×	×	(5)	✓	0.059 / 0.018 s (4)
SatPlan_2004	✓	×	×	×	×	✓	✓	×	×	0.247 – 0.512 s
STAN 4	×	×	×	×	×	×	✓	×	×	0.093 s
TLplan	×	✓	✓	✓	✓	(6)	×	✓	✓	(7)

Abbreviations: Type hierarchies (TH); equality operator (EO); negative preconditions (NP); conditional effects (CE); existential preconditions (EP); optimal (Opt); partial-order planner (POP); action costs (AC); numeric variables (NV); Time to solve test problem (Time). *Notes*: (1) The hsp_s variant took considerably more time (4 min). (2) Can be optimal or not, depending on how it is configured. (3) First time is for optimal configuration; second time for nonoptimal configuration. (4) First time with numeric variables; second time without numeric variables.. (5) Not explicitly supported, but can be easily emulated. (6) Optimal as long as an appropriate control formula is used. (7) We could not create a usable control formula for our test problem, so timing could not be assessed.

ACM Computers in Entertainment, Vol. 5, No. 1, Article 4, Publication date: April 2007.

Figure 68: Planning Benchmarks (Barros & Musse, 2007)

The problem with traditional planning systems is that they are designed to take a holistic approach to solving the problem in the specified domain. The planner will find a solution using all the available agents and actions. In DISE we are using the planner to solve on an 'individual' per-character basis. This means that any plans which provide actions for other characters as a part of the goal solution, other than the currently acting agent need to be removed. This is because the character does not have a mind control like ability to make other characters act in a way that suits their plan (and if they do special control actions need to be created for that character, using the second character as an input parameter). DISE can avoid this problem with two extra steps. In DISE the problem PDDL file is already dynamically generated at each characters planning stage before being passed on to the planner, to allow the initial problem to match the current world state. This allows the currently active character (in this example npc1) to also be passed in as a predicate fact in the init section of the problem PDDL file as:

```
(currentcharacter npc1)
```

The second step is to make sure that every action can only be performed by the currently active character, with any other characters just being used as the patient of the action (see section 3.2.5 for more theory on agents and patients). This is achieved by adding the following extra precondition for every action:

```
(currentcharacter ?x)
```

Every action for DISE should have the agent character of the action as the first action parameter, for example: `?x - character`, which would match this precondition only when npc1 performs the action.

The problem with planning in a dynamic real-time environment is that agents need to deal with incomplete and incorrect information as the world updates. This means that the agents need to revise their plans and knowledge base to cope.

There are two methods to achieve this:

- Execution monitoring and re-planning.
  - Can use classical, sensorless and conditional planning techniques to construct a plan, but it also uses execution monitoring to judge whether the plan has provision for the current situation or needs revising. Re-planning occurs when something goes wrong.
- Continuous Planning.
  - Designed to persist over a lifetime. It can handle unexpected circumstance in the environment, even in the middle of constructing a plan; also the abandonment of goals and the creation of additional goals using goal formulation.

```
Function: Continuous POP Agent(percieved_world_facts)
//returns an Action
Static: plan //a plan, with just [start] and [finish] at the
beginning
Action //defaults to NoOp
Effects[Start] = Update(Effects[Start],percieved_world_facts);
Remove_Flaw(plan) //this could update/create new Action
Return:  Action
```

DISE characters use a custom variant on the former method: Execution monitoring and re-planning, which is processed using the Character Engine. This method allows a more flexible system, which is not dependant on the linked external planner's inclusion the Partial Order Planning (POP) functionality.

5.7 PLAYER ACTION ENGINE

5.7.1 PLAYER-ACTION INTERFACE AND UI DESIGN

Action UI – The action UI will provide the interface for the player to interact with the story's world. It provides a traditional first-person view and allows the player to move with the computer keyboard and look around with the mouse. To carry out an action the player must first left-click on something of interest. After considering a text-based approach to choosing actions, we decided an object based selection method would be more familiar to users and provide a faster and freer flowing experience. When the mouse reticule is moved over an object (or character) that is interactive it will highlight and spin giving instant feedback of the items interactive nature. This interface is tried and tested

and was very widely used in popular classic point and click adventure games such as LucasArts' The Secret of Monkey Island (Lucas Online, 2009). When the object is selected its name will be added to the action phrase and a 3d cloud of verbs will appear around it. Pressing the corresponding labelled keyboard key can choose a verb/action. The action UI is basically a tool to fill in the required parameters for an action, so the next step can vary depending on the requisites for each particular verb. Some parameters are sent automatically such as:

```
Character - player1
Location - player1_location
Character/Item/Dynamic - selected_noun
Location - selected_noun_location
```

For example the action '**trade**' would be carried out as follows:

1. Player1 moves towards another non-player character 'NPC1'.
2. Player1 clicks on character 'NPC1'.
3. Player1 selects 'trade' action from verb cloud UI.

   The action trade has five parameters:

```
(?c1-Character ?c2-Character ?i1-Item ?i2-Item ?l-Location)
```

   The following parameters the computer already knows:

```
?c1=Player1 ?c2=NPC1 ?l=player1_location
```

4. Player1 selects an item for `'?i1'` from a newly created item cloud by pressing the corresponding keyboard key. The selectable list of items is generated from the 'trade' action's preconditions. To trade with another character one item must be yours and the other must be from their inventory (see below). The precondition for `'?i1'` is: `(haveitem ?c1 ?i1)`, so the list will only contain items from player1's inventory.
5. Player1 selects the other item as above, but using precondition: `(haveitem ?c2 ?i2)`, which means that the list will only have items from NPC1's inventory.
6. Now the full action phrase: "Player1 Trade NPC1 item1 (for) item2" can be displayed and executed by pressing the 'return' key. To go back one step the user can press the 'backspace' key or the 'escape' key to quit the action completely move around or choose another interactive object to act on.

In our first prototype action interface we tried the circular menu approach that is used to great effect in games such as Mass Effect (BioWare, 2011); when the player clicks the left mouse button the object is selected and a circular menu will be displayed showing which actions are possible for that object. The action can be selected by highlighting its sector by changing the angle between the mouse cursor and centre of the menu then left-clicking the mouse to confirm the choice.

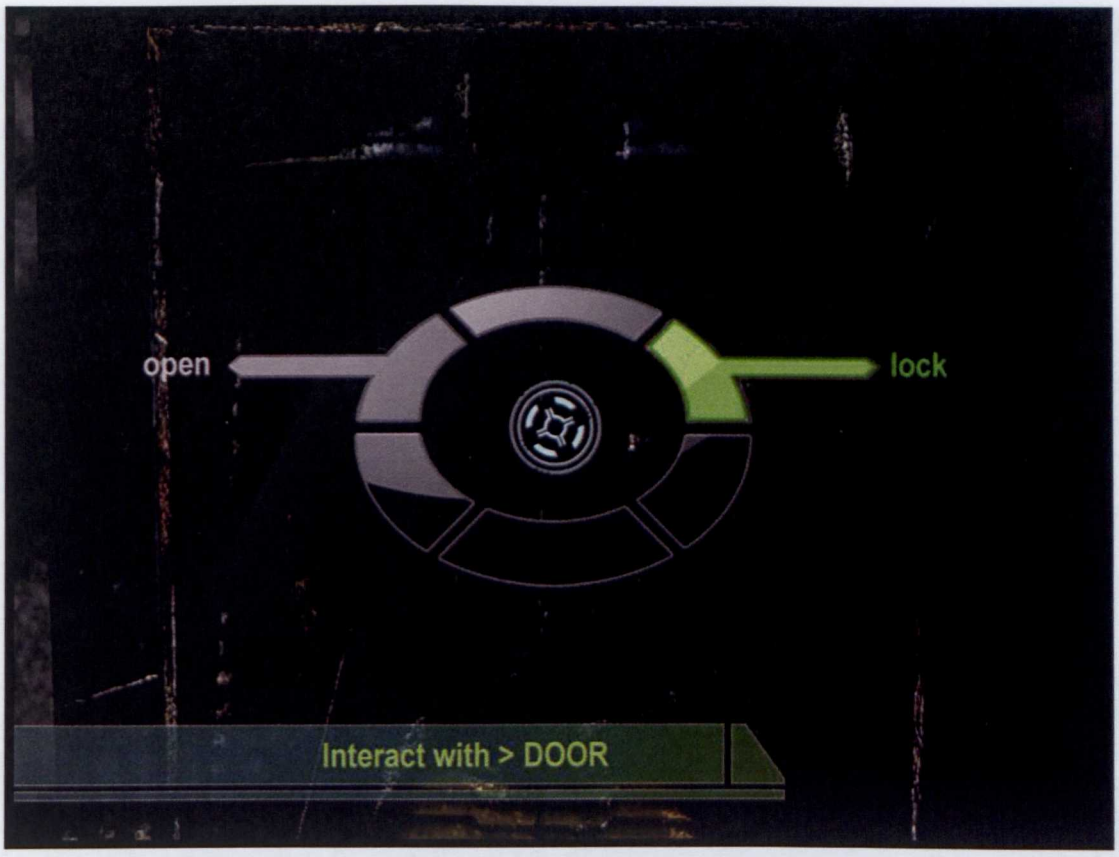


Figure 69: Action Menu Prototype1 – Circle Menu

The second prototype uses the more integrated verb cloud interface (Figure 70), which is similar to the interfaces presented in games such as Heavy Rain (Quantic Dream, 2011) or Dinner Date (Stout Games, 2011). When an interactive object is chosen a cloud of actions rotate around the specified object in the scene. This menu is more immersive as it is not overlaid on top of the game world, but is displayed in perspective 3d around the object of interest. The cloud design is also more adaptable for larger numbers of actions as the text can be scaled and scattered differently depending on numbers, whereas the circular menu is limited to six sectors at a time.

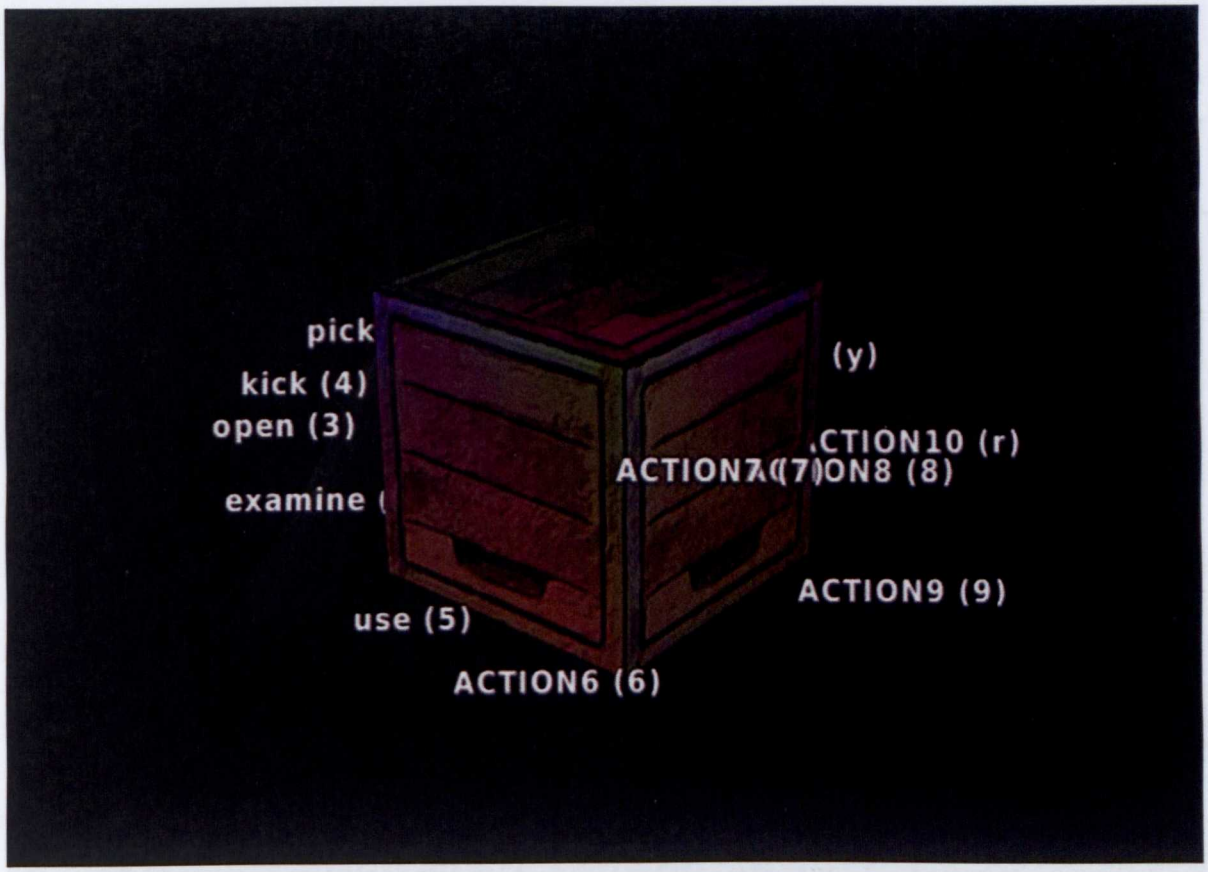


Figure 70: Action Menu Prototype 2 – Cloud Menu

Figure 71 shows a design mock-up of the cloud interface, the action phrase and how the decision making sequence is carried out by the player.

1. Label 1: in the diagram below shows the mouse cursor's state when an interactive object is highlighted, in this case a wooden crate.
2. Label 2: the cloud shows that the 'use' action has been selected and has turned blue by a press of the '5' key.
3. Label 3: this is where the full action phrase can be pieced together and shows the state of the action. The blue word is the currently active one that requires more information. The white words 'player1', 'use' and 'crate' have been filled in just by clicking the crate and selecting 'use' in the first two steps. The 'use' action is also underlined, which shows it is a word that the player has chosen, in this example by pressing the '5' key. If a mistake was made this sentence can be re-edited: the 'escape' key quits completely, going back to the interactive object selection and moving around stage. The 'backspace' key reverts back one step to the previous underlined word or quits to the object selection stage as above if there are no more steps to revert to.
4. Label 4: this is the open inventory button and needs to be clicked to select a noun of the type 'item' to use with the crate.
5. Label 5: shows that player1 is holding the 'crowbar' item, which is

highlighted in blue when it can be used.

6. Label 6: after the crowbar is selected the noun will be placed into the action phrase, making it complete. Player1 can hit return to carry out this action or press the 'esc' or 'backspace' keys to go back and change it.

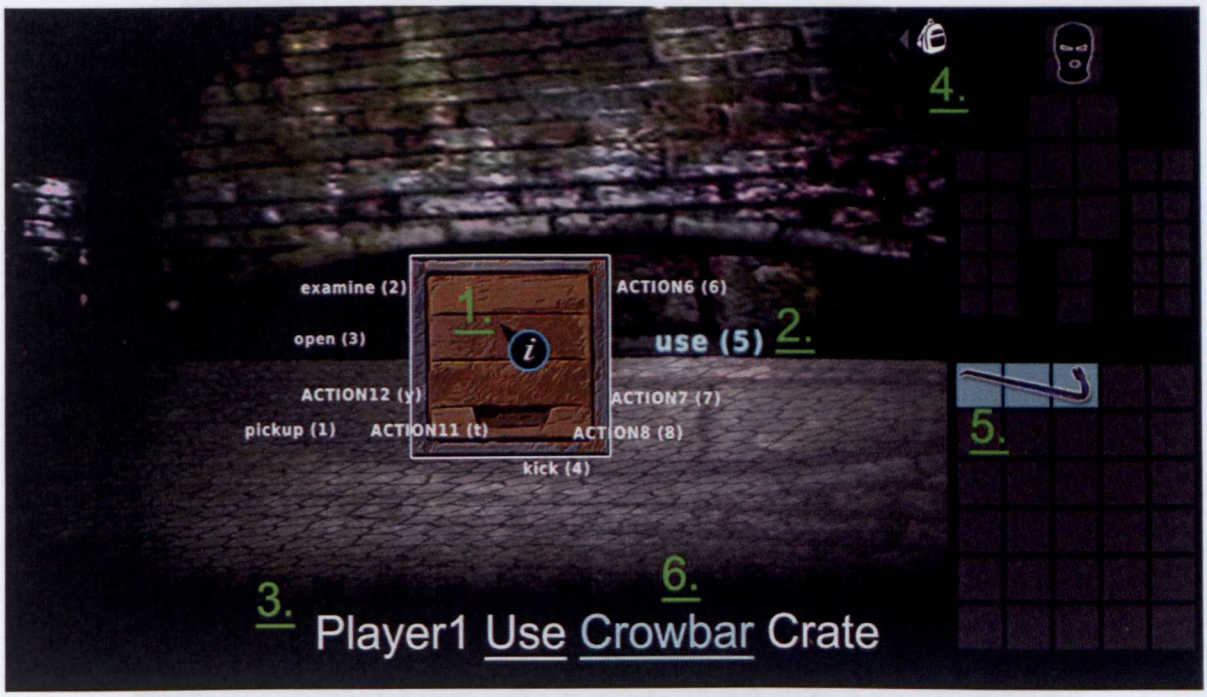


Figure 71: DISE UI Action Sequence

So the summary of player input with the following state of the action phrase for the above example is:

1. Walk to crate and click on it with the mouse: `player1, blank, blank, crate.`
2. Press the '5' key to select 'use': `player1, use, blank, crate.`
3. Open inventory by clicking the bag button and select the crowbar: `player1, use, crowbar, crate.`

Inventory – The inventory contains all the items a character is holding and wearing on their person. We will use a similar system as many classic adventure games have successfully used in the past (See Figure 72). The inventory screen can be called up to interact with held items, for example wear–hat or drop–bag. The player has the following inventory sections:

- Pockets/Bags Space – the story author can customise how much item storage capacity the player has in block units. Each item has a size in blocks and a particular pattern/shape that will add an item management aspect to the story. This inventory size and space limitation is of course optional and can be removed altogether by the story author for their story, giving infinite

storage space for items if that aspect is not important to the story using the "Limit Inventory Size=false" check box in the editor.

- Clothes and Accessories – this section will show what items of clothing a character is wearing, such as shirts, hats, glasses, gloves, watches and jewellery.

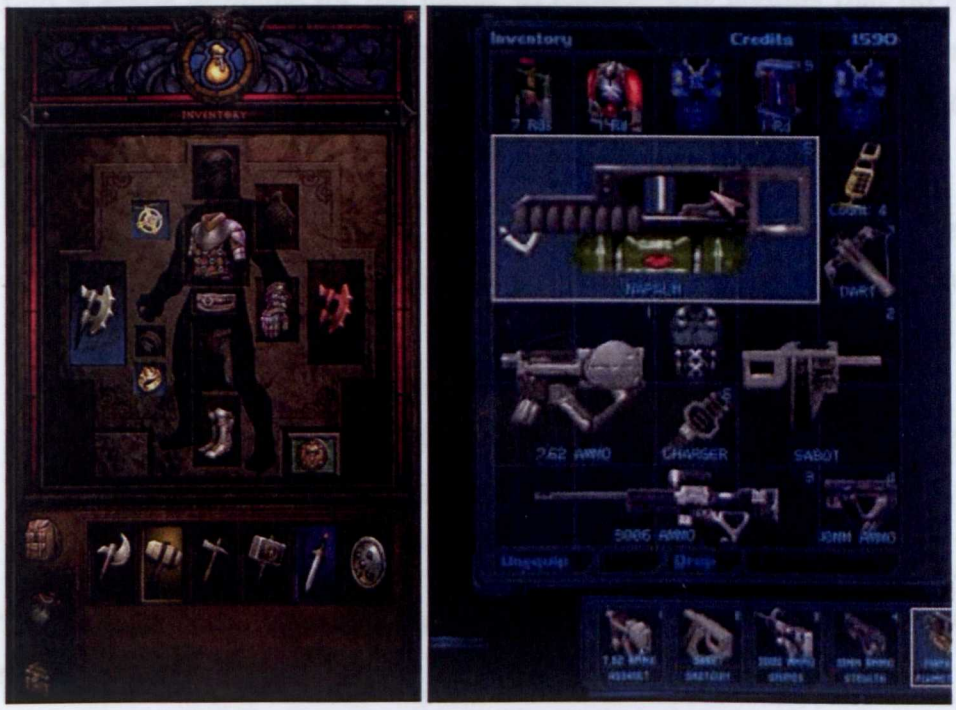


Figure 72: Inventory screens from Diablo 3 and Deus Ex

5.7.2 INVERSE PARSER

To streamline the action user interface and only allow action choices that are a possibility in the given context we will tie an inverse parser into the planning data. This will mean that the user can never pick an impossible action and the lack of error messages will maintain immersion in the story. When an object or character is selected with the mouse the action menu is automatically filled with plausible actions. This is achieved by inversely parsing the parameters and their preconditions and only selecting actions that have at least one choice for all the parameters with the precondition equalling true. Actions parameters are listed as follows in a similar to Brémond's ROLES & PROCESSES 3.2.5:

- 1st Parameter: Character – The Agent of the action.
- 2nd Parameter: Character – The patient acted on OR Item/Dynamic – The thing acted on.
- Last Parameter: Location – The location of the agent and is usually required to be the same location for the patient or thing acted on.

The parameters in-between can vary depending on the action.

In the 'trade' action example above, when player1 clicks on NPC1, extra steps are taken before step 4 by the action engine behind the scenes to check ahead and populate the list of verbs. As stated before the computer already knows the parameters: `?c1=Player1 ?c2=NPC1 ?l=player1_location`, so in this case if both player1 and NPC1 have an item, then the preconditions are true and this action is made available for the player to choose. Any actions that do not have: 'Type – Character' for one of their parameters (p) between p=2 to p=n can be instantly negated. The inverse parser takes the following steps to narrow down the search:

1. Check one of the parameter types matches with the clicked 'object/thing' (ignoring p=1 which is the actor of the action, in this case player1).
2. Fill in parameters that we already know.
3. Create lists for ones that are empty matching the parameter types.
4. For each thing in list, check that it passes the preconditions.
5. List the action as available in the menu.

For example consider the following actions (we have omitted their effects as we are not concerned with that yet):

```
(:action wear
:parameters (?x - character ?y - item)
:precondition (and (haveitem ?x ?y)(iswearable ?y))
)

(:action read
:parameters (?x - character ?y - dynamic ?z - location)
:precondition (and (at ?x ?z)(at ?y ?z)(isreadable ?y))
)

(:action store
:parameters (?x - character ?y - item ?z - dynamic ?w - location)
:precondition (and (haveitem ?x ?y)(at ?x ?w)(at ?z ?w)(isstorage
?z))
)

(:objects
shelf - dynamic
book - item
room - location
)

(:init
(at shelf room)(at player1 room)
(isstorage shelf)
(isreadable book)
(hasitem player1 book)
)
```

If the clicked object is 'shelf – Dynamic' and the actions in the current scene were wear, hang-up and read, then:

- Wear: doesn't contain a dynamic, so would be rejected at this stage.
- Read: has dynamic, so continues to next round.
- Store: has dynamic, so continues to next round.

Now it is time to fill in the parameters that we know and drill down to the preconditions.

- Read
  - We know that character x = player1, dynamic y = shelf and location z = the player's current location, so have all the required parameters.
  - Checking the preconditions will flag up that shelf is not readable, so fails at this stage.
- Store
  - We know that character x = player1, dynamic z = shelf and location w = the player's current location.
  - The computer has to query item y and its preconditions to proceed.
  - The selected dynamic – shelf is storage and the character and shelf are at the same location so these return true.
  - If a player has any item y then this action is selectable.
  - If this action is chosen, a list of items is generated all needing:

```
(hasitem player1 item) = true
```

  - As book is the only item in the player's possession, the final action would be: **Player1 Store Book (on) Shelf (in) Room**.

This technique makes sure the player is not getting negative feedback for any actions that they select, but can take extra processing time to populate action menus, which can increase depending on how many actions are being considered in the particular scene and how many match up at the earlier stages.

The table in Figure 73 shows what actions would be appropriate to display for each example object when it is selected with the mouse cursor.

	Verbs to choose						
Object Selected	Store	Trade	Pick up	Open	Insult	Tell Secret	Wear
Shelf	X						
Jack		X			X	X	
Jill		X			X	X	
Shoe		X	X				X
Door				X			
Bag			X	X			
Coin			X				
Gun			X	X			

Figure 73: Object/Verb Linkage Table

5.8 CHARACTER ENGINE

The non-player characters (NPCs) are an important aspect of the story world as their behaviour and interactions with the player can generate narrative. In this section we explain the character engine, which is responsible for sequencing their turns and planning stages; firstly with a brief overview. Following are that the action contents of a characters class: Goal, Action and Execution Lists. The next section describes in depth how the characters logical states are processed in a character engine sequencing turn, to process planning, re-planning and execution of actions stored in execution lists, along with the pre-condition checks needed to see if actions are still possible at their follow up time step. The personalities of characters are explained in the next section, along with how they are integrated into the planner. Finally the last two sections cover how characters are given new goals using reactions and choice formulae and how characters are created using the Character Editor Panel.

5.8.1 CHARACTER ENGINE OVERVIEW

The character engine handles the behavioural control of each character. It holds the array of story characters and is responsible for updating their turn logic to process plans, goals and actions. Each character holds a list of actions it can perform and their current goal/s (thus defining their role), along with the action plan list, action execution list, current goal, current action, number of re-plan attempts and wondering state. The character engine acts like an inference engine in an expert system, deriving answers for each character using the fact database, the character's control variables and the instance of the planner that is dynamically created at the plan and re-plan stages.

The character engine contains a list of NPCs (non-player characters) currently in the story. These characters hold data for their goals, available actions, lists of actions for future plans and actions to be executed straight away. The relationship can of character engine, the characters and the goal and action data can be seen in Figure 74. These are described in greater detail in the next section 5.8.2 GOAL, ACTION AND EXECUTION LISTS.

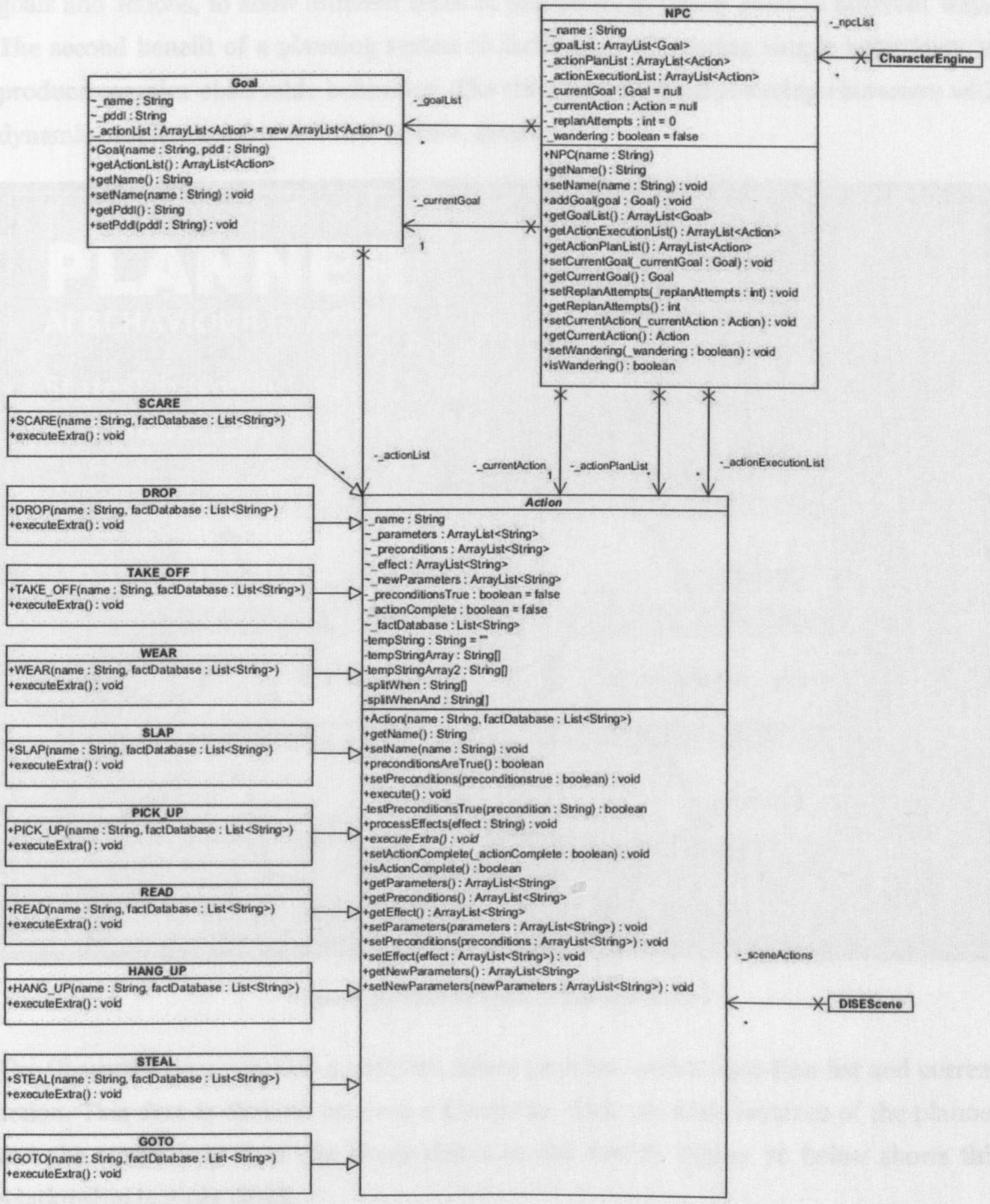


Figure 74: Detailed Character Engine Class Diagram

5.8.2 GOAL, ACTION AND EXECUTION LISTS

Characters interact with both the world and the planner via their goal, action and execution lists. Each character has a state, set either to their currently active action or wandering mode if no actions can be derived from their current goal list. When the story starts characters should be given at least one goal to process by the story author, unless they are required to wander around until a goal is given later in a specific scene using 'Story Mods', which are explained in more detail in section 5.5.2. Orkin states that using goals and planning has three main benefits "The first benefit is the ability to decouple

goals and actions, to allow different types of characters to satisfy goals in different ways. The second benefit of a planning system is facilitation of layering simple behaviours to produce complex observable behaviour. The third benefit is empowering characters with dynamic problem solving abilities" (Orkin, 2006).

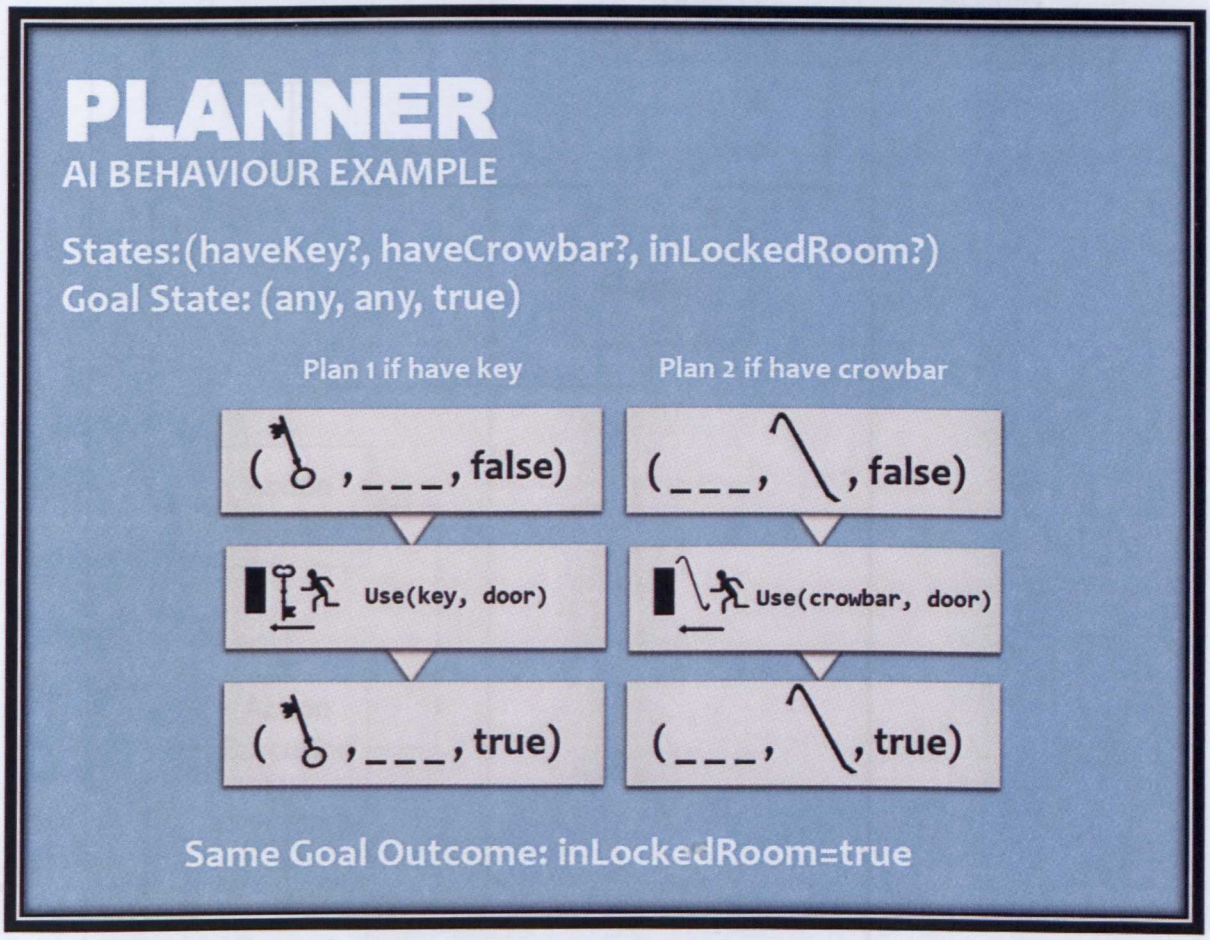


Figure 75: Plan to Enter a Locked Room

The Character class contains a goal list, action plan list, action execution list and current action. This data is chained between a Character, their personal instance of the planner and the world state data (for every object in the world). Figure 76 below shows this relationship in more detail.

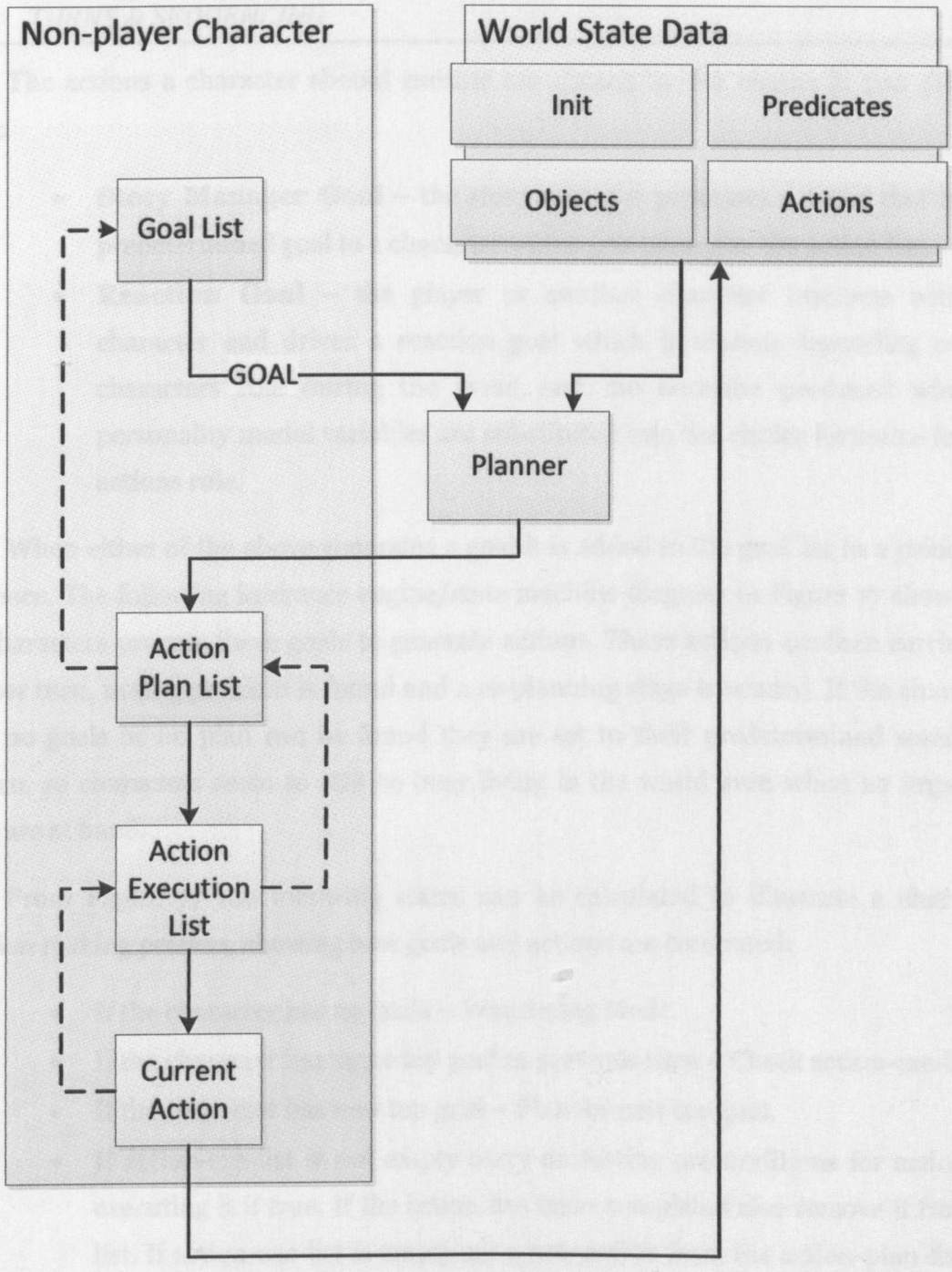


Figure 76: Character, World and Planner Interactions

The actions a character should execute are chosen by the engine in two different ways:

- **Story Manager Goal** – the story manager processes a scene that feeds a predetermined goal to a character which gets passed to the action list.
- **Reaction Goal** – the player or another character interacts with this character and drives a reaction goal which is chosen depending on this characters role during the event and the outcome produced when its personality model variables are substituted into the choice formulae for that actions role.

When either of the above generates a goal it is added to the goal list in a prioritised sequence. The following inference engine/state machine diagram in Figure 77 shows how the characters process these goals to generate actions. These actions are then carried out one per turn, until a problem is found and a re-planning stage is needed. If the characters have no goals or no plan can be found they are set to their predetermined wandering pattern, so characters seem to still be busy living in the world even when no important tasks are at hand.

From Figure 77 the following states can be calculated to illustrate a characters decision making process, showing how goals and actions are computed:

- If the character has no goals – Wandering Mode.
- If the character has same top goal as previous turn – Check action-exe-list.
- If the character has new top goal – Plan for new top goal.
- If action-exe-list is not empty carry on testing preconditions for action and executing it if true. If the action has been completed also remove it from the list. If action-exe-list is empty get a new action from the action-plan-list, test preconditions and move to execute if true.
- If action-plan-list is empty then all the actions for current goal are complete, so get new goal from goal-list.
- If preconditions are now false, a new plan is generated for the current goal.
- If a plan cannot be found then an attempt at re-planning is made, if this fails the character is temporarily put into wandering mode for this turn.
- If no plan can be found after 10 turns the goal is demoted to the bottom of the goal-list, so that other goals can be given priority instead.

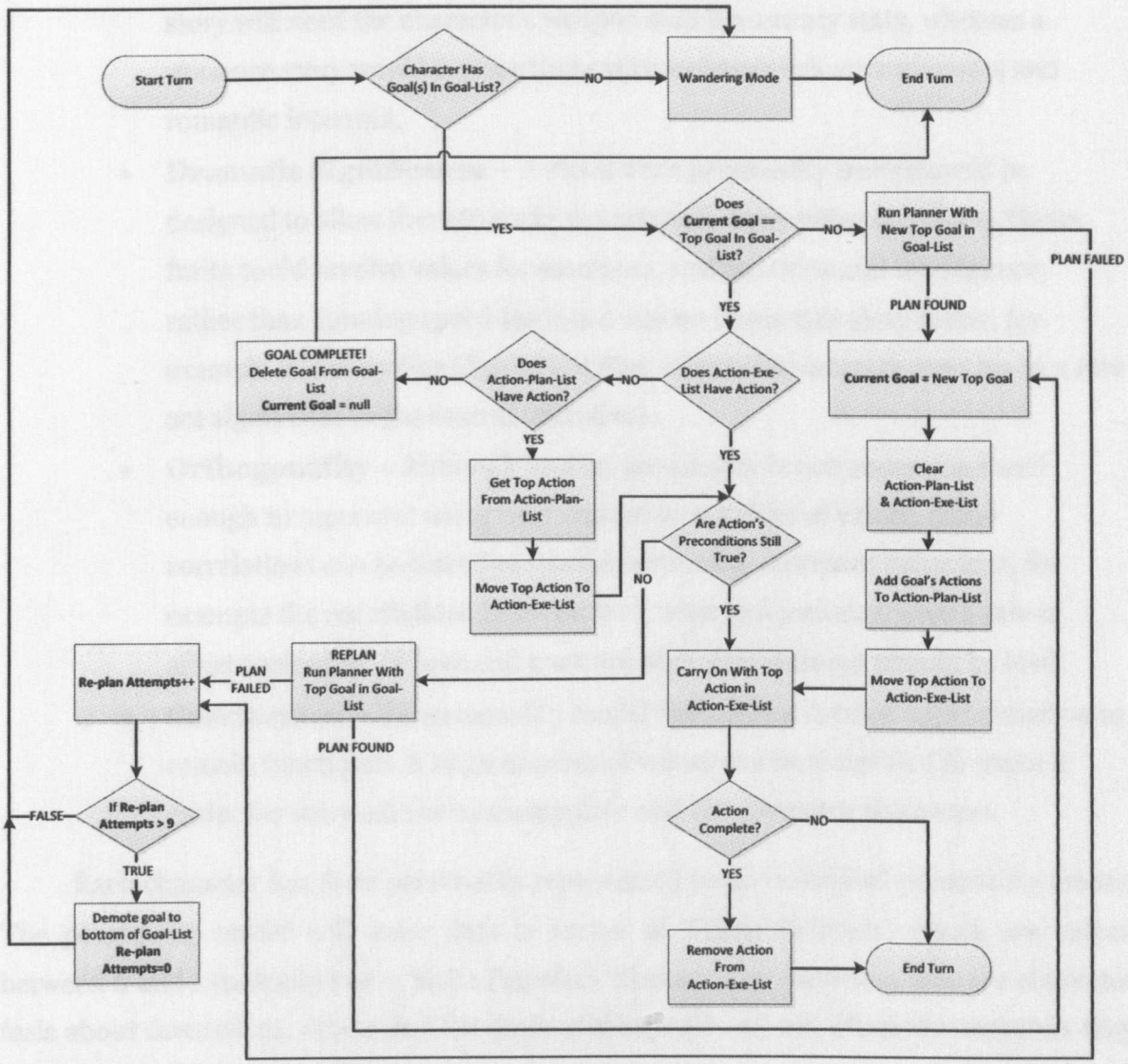


Figure 77: Character Planning Turns

5.8.4 PERSONALITY MODELLING

Characters play a big role in stories and interactive software developers have been refining the behaviour of non-player characters over the last two decades to make them more dynamic and engaging. This had increased the scope of the personality models, making them more detailed with many variables representing different aspects. Many scientific personality models have been developed in academic fields such as psychology (including the Biospheric, HEXACO, Hypostatic and Two-factor models of personality), but these meet a different set of requirements than Interactive Storytelling, so can only be used as a guide.

In his paper (Crawford, Personality Modelling for Interactive Storytelling, 2004) Crawford lists these requirements as:

- **Behavioural Completeness** – Any personality model must address all possible NPC behaviours associated with the story's genre. E.g. a shooting

story will need the character's weapon skill & accuracy stats, whereas a romance story would replace these with a character's attractiveness and romantic interests.

- **Dramatic Significance** – A character's personality traits should be designed to allow them to make dramatically interesting decisions. These traits could involve values for emotions, relationships and intelligence; rather than running speed (unless it was an important story factor, for example in a story like Chariots of Fire, where characters competing in a race are significant to the central narrative).
- **Orthogonality** – Although human personality is not understood well enough to represent using neat vectors or matrices of values, some correlations can be linked to more dynamically represent behaviour, for example the correlations between love, trust and jealousy, where values affect each other (if love and trust are high then jealousy should be low).
- **Conciseness** – the personality model should be a concise approximation to remain functional. A large number of values can be simplified to make it easier for story editors to manipulate and the computer to process.

Each character has their personality represented by an individual personality model. The personality model will store data in arrays of 'Fuzzy Switches', which are values between 0 and 1 (unipolar) or -1 and 1 (bipolar). These values represent how the character feels about themselves, others and the state of the world and will affect the decisions they make and the actions that they choose.

The variable types that the personality model needs to include are:

- **First person variables** – one dimensional array of personal character traits, moods and moral values. For example:

```
Happiness(character1) = 0.5f.
Laziness(character1) = 1f.
```

- **Second person variables** – two dimensional arrays describing character relationships. They contain both a subject and a focus character and depict the subject's perception of the focus character. In this example character 1 thinks that character 2 is honest but is not attracted to them.

```
PercievedHonesty(character1, character2) = 1.0f;
PercievedAttractiveness(character1, character2) = 0.0f;
```

- **Third person variables** – three dimensional arrays describing one third-party character's perception of another's feelings towards someone else. In this case character 3 guesses the opposite for the perceived honesty of character 1 for character 2 from the example above, but perceives correctly

that character 1 is not attracted to 2.

```
PerPercievedHonesty(character3, character1, character2) =
0.0f;
PerPercievedAttractiveness(character3, character1,
character2) = 0.0f;
```

- **Accordance Variables** – these describe a characters willingness to perceive high values. Accordance describes character traits such as gullibility or suspiciousness (which is determined by a high 'accordance to honesty' for the former and a low value for the latter). In this example character 1 wants to believe that people are honest but is very picky about attractiveness.

```
AccordanceHonesty(character1) = 0.8f;
AccordanceAttractiveness(character1) = 0.1f;
```

Crawford's Erasmatron system (which later evolved into Storytron, described in section 3.1.19) uses the following personality model, which could be easily implemented in the DISE characters:

- First-order variables

```
Honest
Virtuous
Powerful
Intelligent
Attractive
```

- Accordance variables

```
AccordHonesty
AccordVirtue
AccordPower
AccordIntelligence
AccordAttractive
```

- Second-order variables

```
PerHonest
PerVirtue
PerPowerful
PerIntelligent
PerAttractive
```

- Third-order variables

```
PerPerHonest
PerPerVirtue
PerPerPowerful
PerPerIntelligent
PerPerAttractive
```

- Moods (bi-polar +/-)

```
Passion/Disgust
Joy/Sadness
Anger/Fear
```

In DISE the personality model only limits story authors to the five categories: First-

order, Accordance, Second-order, Third-order and Mood variables. Within these fixed categories any new variables can be added to the models master list by a story author using the editor, as long as they have a unique identifier name. For example if the story was in the detective genre, the author may want to add a variable for 'detection' in the characters' first-order variables list. If an NPC has a high detection value they could be programmed in the current scene to inform the player that they have noticed a clue to progress the story.

5.8.5 NEW GOAL GENERATION

For every action that is carried out we want a set of reactive goals to be available along with the PDDL effects state change rules, if deemed necessary for that action. To provide this the effect section will also contain additional rules that describe how non-player characters (NPC) should react to a player's actions and determine which goals they will choose. The NPCs will be sequenced using the concept of roles. For each different role there are four main elements to consider (*Figure 78*). A role can be anything that a story builder defines in a verb/action and more roles and reactions could be added later if they are needed. Some actions have a direct subject to them that can be easily defined; other roles could be characters that passively observed the action and want to intervene.

Effect:

NPC Roles:

1. **Rules to describe which character should react to the event and which role they fit into.**
2. **The Modifiers that change the personality variables of the character in the specific role, which reflects how they feel about the current action and how it physically affected them.**
3. **A list of Reaction Goals to choose from and act out.**
4. **A Choice Formula that allows the character to choose the most suitable verb from the aforementioned list according to their updated personality model.**

Example:

```
Role:
• Receiver - Character
Modify:
• if Likes(Receiver, Item) then Likes(Receiver, Giver) + 0.1f
  * Amount_Receiver_Likes_Item
React:
• if Likes(Receiver, Item) then Thank(Receiver, Giver)
• if Likes(Receiver, Item) And Likes(Receiver, Giver) > 0.8f
  And In_Relationship(Receiver, Giver) then Kiss(Receiver,
  Giver)
• if !Likes(Receiver, Item) And !Likes(Receiver, Giver) then
  Give(Receiver, Giver, Item)
Role:
• Witness - Character
• Witness != Giver And Witness != Receiver And Witness See Give
Modify:
• Perceived_Generosity(Witness, Giver) + 0.1f * Item_Value
• If Likes(Witness, Item) And Jealous_Person(Witness) then
  Jealousy(Witness, Receiver) + 0.1f * Likes(Witness, Item)
• If Likes(Witness , Giver) And Jealous_Person(Witness) then
  Jealousy(Witness, Receiver) + 0.1f * Likes(Witness, Giver)
React:
• If Jealousy(Witness, Receiver)>0.9f then
  Snide_Comment(Witness, Receiver)
```

Figure 78: Effect of NPC Roles, Modifications and Reactions

Figure 79 below shows a sequence of events and roles allocated to both the 'subject' character and the 'witnesses' of these events. If we take the example above with Player 1 being the 'giver' and CPU1 the 'receiver' of a gift using the event 'gives' the first role to be cast is the 'receiver' character. This will then allow them to process their feelings and choose a reaction to the event based on the variables of their personality model cross-referenced with the 'gift' and 'giver'. The other two CPU characters are in line of sight of the event and become 'witnesses', so get their turn next to process this event. Depending on the value of the object, the amount the 'witness' likes the 'giver' and their tendency to be jealous, they will make a snide comment or just have an increased perception of the 'givers' generosity. The last character can see the event but is prioritised to ignore roles for this event for now as they are busy, so do not get affected by the scene.

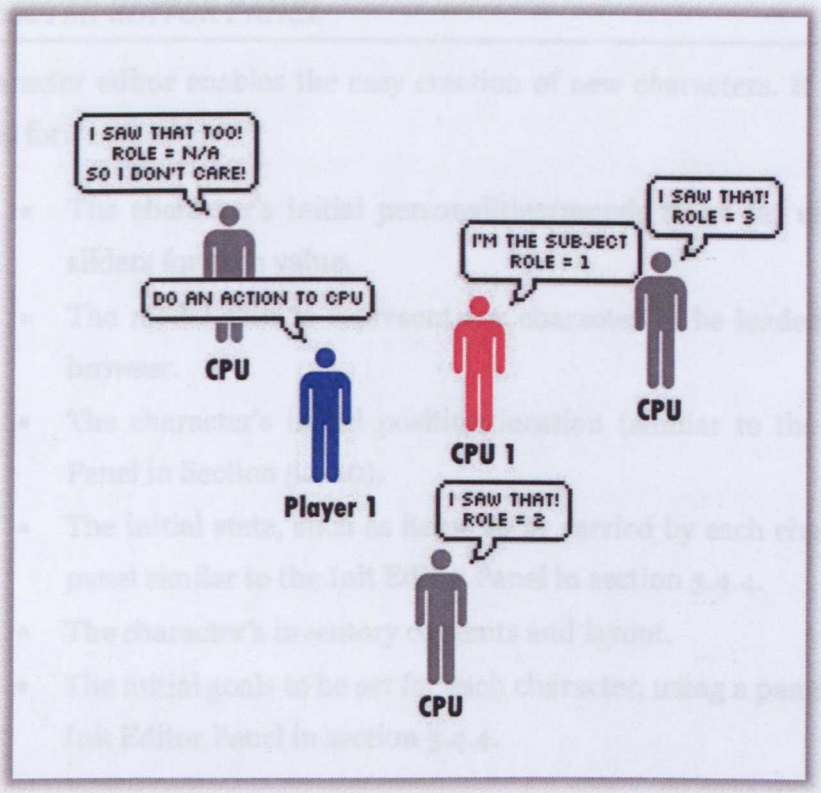


Figure 79: Character Role Sequence

5.8.6 CHARACTER EDITOR PANEL

The character editor enables the easy creation of new characters. It has data entry user interfaces for:

- The character's initial personalities/moods to be set using individual sliders for each value.
- The model skin to represent the character to be loaded using the file browser.
- The character's initial position/location (similar to the Object Editor Panel in Section 5.4.10).
- The initial state, such as items to be carried by each character, using a panel similar to the Init Editor Panel in section 5.4.4.
- The character's inventory contents and layout.
- The initial goals to be set for each character, using a panel similar to the Init Editor Panel in section 5.4.4.

5.8.7 CHARISMA CHARACTER ANIMATION SYSTEM

Charisma is the character animation system in DISE which controls the detailed animated characters. Charisma combines MPEG-4 compliant facial animation streams with skeletal-based animations, in order to improve the realism and smoothness of the movements within the animation, as well as simplifying the animation pipeline. We use a novel approach that extends and improves Garland's Quadric-Based Surface algorithm, by providing a solution that is both MPEG-4 facial animation (MPEG-4 FA) model compliant and maintains the model feature points whilst conserving its realism as much as possible, improving the performance of the MPEG-4 facial animation implemented in Charisma (Carter, Cooper, El Rhalibi, Merabti, & Price, 2010).

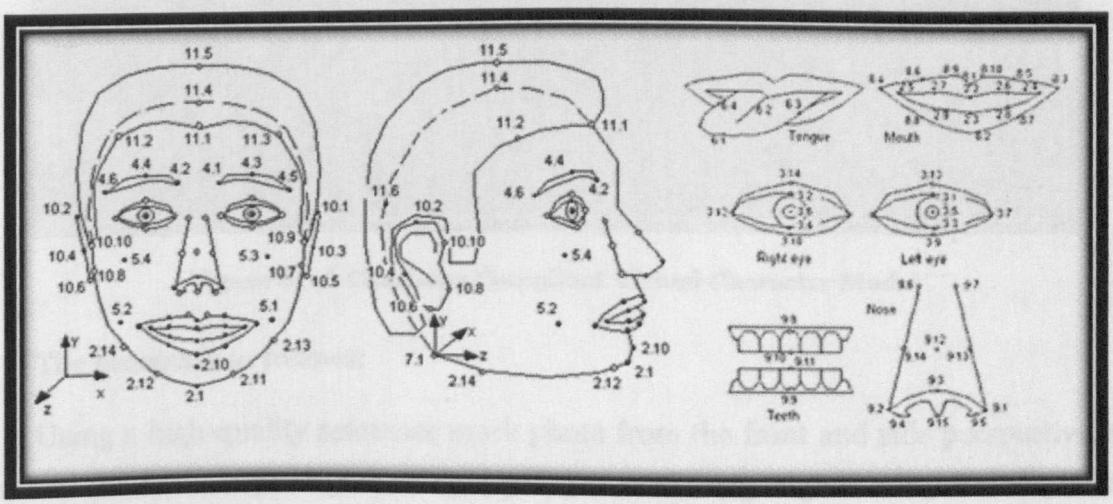


Figure 80: MPEG-4 Feature Points

The MPEG-4 FA provides a concrete specification for representing a virtual head as a set of 84 features points that define areas of the face which can be manipulated through a set of 68 parameters values (FAPS), each applying a scalar transformation along a given axis to a particular feature point (Figure 80). These feature points are key elements in a human facial model for realistic animation and expressivity.

In order to simplify the animation process and reduce the amount of code required to integrate a smooth MPEG-4 compliant animation system, we have developed a system where the artist is tasked with defining the structure of feature points and the corresponding vertices to be manipulated. In order to construct a head model which is compliant with the Charisma API and with sufficient detail to display the required amount of emotional response, we use Autodesk 3D Studio Max 2010, due to its industry-standard status. Figure 81 shows an example of a Charisma compliant virtual character model Gabrielle; textured and rigged with the skeletal structure which corresponds to the FDP specification. The weight influences of the bones are highlighted, increasing in strength from blue through yellow, to red.

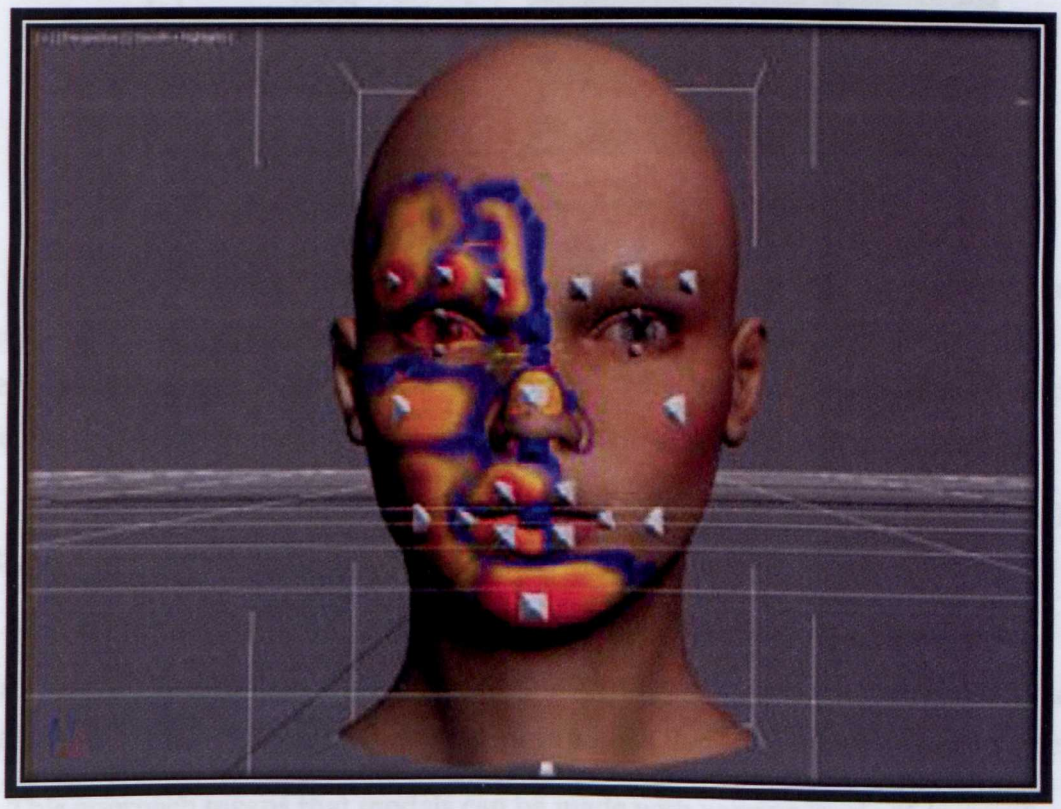


Figure 81: A Charisma Compliant Virtual Character Model

The process is as follows:

Using a high-quality reference stock photo from the front and side perspective, the shape of the head is produced using a polygon modelling edge-cloning technique. The mesh is then unwrapped and textured using the reference photographs as a base. To rig the head for animation, a framework of bones must be created. The position of the bones

is determined from the feature points taken from the MPEG-4 specification (Forchheimer, Pandzic, & Pakstas, 2002). Once the bones are positioned on the face, they are labelled in accordance to the FDP feature point they are representing (e.g. 3.1 becomes bone _3_1). In order to eliminate the need for FATs, we have substituted them with their skeletal equivalent: blended bone weights. Figure 81 illustrates a modelled, textured, bone positioned and weighted face model. The FAPU definition areas, used to calibrate the model and define the unit transformation effect which each FAP has on its associated feature point (bone). As seen in the image below, in our technique, these are determined intrinsically from the model itself, requiring no additional meta-data or data processing.

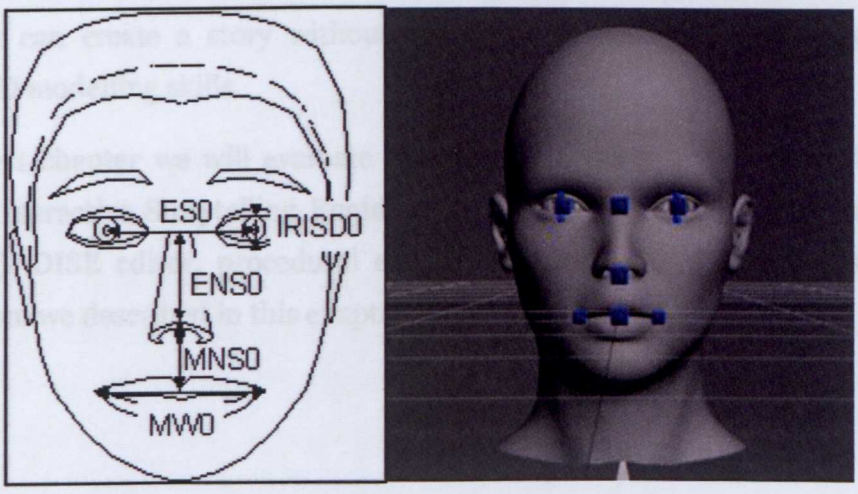


Figure 82: FAPU Definition Areas in Charisma

The model is exported into the ubiquitous standard model interchange format, Collada. This format preserves all the data, whilst providing interoperability with a wide range of additional modelling and texturing tools. This approach also allows all steps for producing a Charisma-compliant model to be completed inside a single application.

Our pipeline also allows the bones to be manipulated inside the editor, in order to rapidly test and incrementally improve the realism of the model.

The MPEG-4 calibration step is performed directly from the model's data. We utilize the skeletal structure of the model to determine the FAPUs. Additional marker bones in between the eyes and the fulcrum were added. Figure 4 illustrates this concept.

This approach means that models can be made with varying levels of expressiveness. The model is coloured using a multi-texturing approach. When the model is loaded into the renderer, standard OpenGL lighting can make it look lifeless. To combat this we use a GLSL-based hardware lighting Shader, in combination with normal and specular maps. These maps are stored in the form of two additional textures, which are combined with the standard diffuse texture. The GLSL Shader is executed directly on GPU, performing all transforms and lighting in a parallel manner.

5.9 CHAPTER SUMMARY

In this chapter we gave a detailed account of how we implemented our design and built the DISE framework. The basic structure of DISE was split into player and editor components then further categorised into smaller interlinked sub-systems, which each have a specific role. We explained these various systems and data structures and provided screenshots of the user interface and 3D graphics where appropriate. From these details we hope the reader can gain a deeper understanding of our approach to building an Interactive Storytelling system which places the player in the centre of the story, giving them choices which affect the narrative outcome. Our section on editing tools also explains how an author can create a story without needing advance syntactical programming knowledge or 3D modelling skills.

In the next chapter we will evaluate the individual parts of the DISE framework including our Interactive Storytelling Engine (which includes the Character Engine and Story Manager), DISE editor, procedural editing tools and finally the Charisma facial animation system we described in this chapter.

"Telling stories is as basic to human beings as eating. More so, in fact, for while food makes us live, stories are what make our lives worth living (Kearney, 2002).

6 DISE FRAMEWORK EVALUATION

In this section we discuss the evaluation of the DISE framework. We will describe the results of experiments that provide evidence in support of this thesis. We chose criteria and experiments that will emphasise either the proof-of-concept (demonstrating the viability of a method/technique) or efficiency (demonstrating that a method/technique provides better performance than those that exist).

The analysis of Interactive Storytelling software is a difficult task as it is hard to define quantitative methods for evaluating the quality of an interactive story, some measures could be taken from the game frame rate and calculation/running speed also the number of possible combinations of actions and effects. Qualitative approaches to story evaluation can focus on which choices players make in game and how much they feel they had an effect on the story world. This could involve several playthroughs of one story, purposefully behaving in a different manner and making alternate choices to previous sessions.

Usability and user-case study tests would involve probing users experience as regard to story quality and how it compares to say a linearly controlled story. The usability evaluation will focus on the story frameworks editors and will consider the main human computer interface usability practice areas: performance, accuracy, recall and emotional response.

We can gain some amounts of quantitative data by measuring the performance and scalability of DISE with an increasing number of active characters, measuring the execution and planning times and the amount of memory usage which is detailed in the next section. Although this does not measure the quality of the story it can measure DISE's ability to cope with multiple characters, its real-time capabilities, its range of supported hardware and range of actions/behaviour which are important for rich storytelling.

6.1 STORYTELLING ENGINE EVALUATION

6.1.1 BENCHMARKING FICTION

Benchmarking has been used for many years to comparatively study the efficiency or standard of different media using a control subject. Interactive storytelling can be represented in many different forms, with different requirements, making the process of benchmarking difficult. Dena et al (Dena, Douglass, & Marino, 2005) research benchmarking for interactive fiction and e-literature and define the term 'benchfic' as: "A benchmark fiction, or 'benchfic', is an e-lit adaptation for the purposes of comparing media. The term 'benchmark' here is playfully repurposed from the fields of computer science and strategic management in order to emphasize the focus on utility and standards. While 'benchmark' originated as a surveying term for a point of reference, in contemporary computer science, 'benchmarking' has come to mean the execution of a software test in order to ascertain the relative performance of underlying hardware." Their research mentions 'Cloak of Darkness' a benchmark for interactive media that focuses on similar behaviour and user experience just as much as story consistence. Dena et al (Dena, Douglass, & Marino, 2005) evaluate 'Cloak of Darkness as "our most sophisticated example of a collection of adaptations proceeding from a specification towards a concept".

The "Cloak of Darkness" specification is as stated on Firth's website (Firth, 2010): "The various implementations have been made as similar as possible. That is, things like object names and room descriptions should be identical, and the general flow of the game should be pretty comparable. Having said that, the games are implemented using the native capabilities of the various systems [...] The target is to write naturally and simply, while sticking as closely as possible to the goal of making the games directly equivalent. There are just three rooms and three objects:

The **Foyer** of the Opera House is where the game begins. This empty room has doors to the south and west, also an unusable exit to the north. There is nobody else around.

The **Bar** lies south of the **Foyer**, and is initially unlit. Trying to do anything other than moving north or west; results in a warning message about "disturbing things in the dark".

On the wall of the **Cloakroom**, to the west of the **Foyer**, is fixed a small brass **hook**.

Taking an inventory of possessions reveals that the player is wearing a black velvet **cloak** which, upon examination, is found to be light-absorbent. The player can drop the **cloak** on the floor of the **Cloakroom** or, better, put it on the **hook**.

Returning to the **Bar** without the **cloak** reveals that the room is now lit. A **message** is scratched in the sawdust on the floor.

- The **message** reads "You have won" or "You have lost", depending on how much it was disturbed by the player while the room was dark.
- The act of reading the **message** ends the game."

Although this benchmark is designed for text based interactive fiction, it will be a good test for the DISE action engine, planner and story editor (Figure 83). If the objects and actions can be defined along with scenes to trigger messages to the player, destroy the sawdust and control the lighting of the room then DISE will be considered versatile enough to create any adventure.

This test could also be extended to involve AI characters with their own goals, to evaluate multi-pass-planning and extra scenes/actions to create multiple ways of solving the puzzle, with more potential outcomes, for example: we could include a foolish character who stumbles around in the dark causing a lose condition, unless you talk to him or give him an item. Another test could substitute the player for a computer AI, allowing them to plan and solve the puzzle alone with the goal of reading the message.

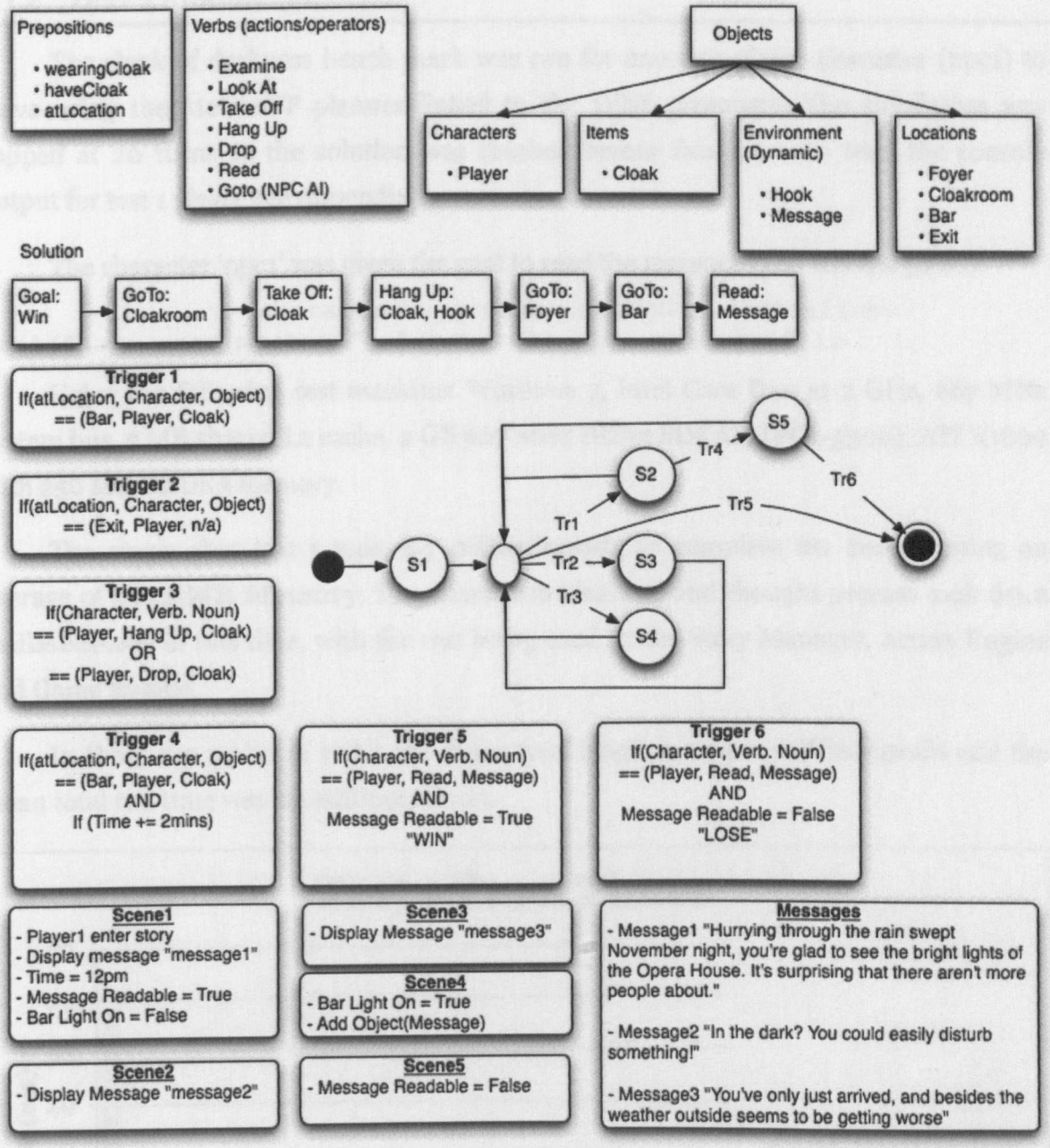


Figure 83: Example of Scene Structure in Cloak of Darkness DISE Implementation

The cloak of darkness bench mark was run for one non-player character (npc1) to solve using the Metric-FF planner linked to the DISE prototype. The simulation was capped at 20 turns as the solution was reached before this time. To read the console output for test 1 please see Appendix 1.

The character 'npc1' was given the goal to read the message (and win the game):

```
_characterEngine.getLastNPC().addGoal(new
Goal("have read message", "(haveread npc1 message)"));
```

Using the following test machine: Windows 7, Intel Core Duo at 2 GHz, 667 MHz system bus, 2 MB shared L2 cache, 2 GB 667 MHz DDR2 SDRAM (PC2-5300), ATI X1600 with 256 MB GDDR3 memory.

The single shot test 1 took **84 milliseconds** to complete **20 turns**, using an average of **47.95MB Memory**. The characters planning and thought process took **69.2 milliseconds** of this time, with the rest being used by the Story Manager, Action Engine and Game Engine.

In the 5 run series of test 1 the mean turn length was **4.1 milliseconds** and the mean total run time was **82 milliseconds.**

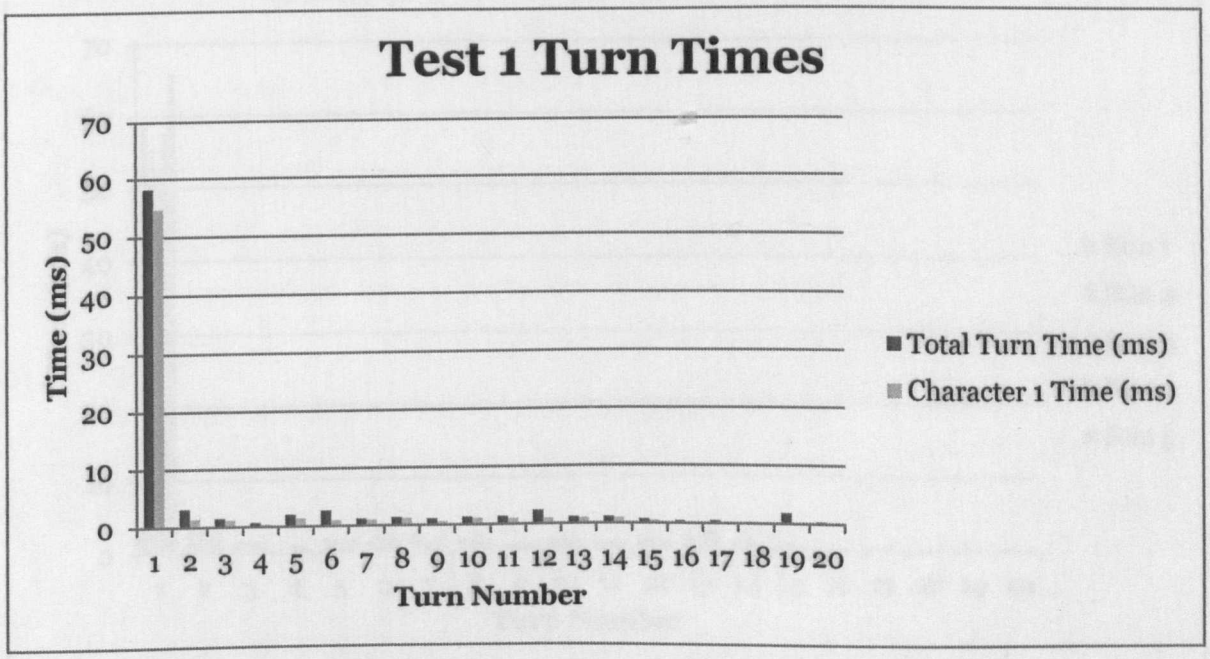


Figure 84: A Graph Showing the DISE CoD Test 1 Turn Times

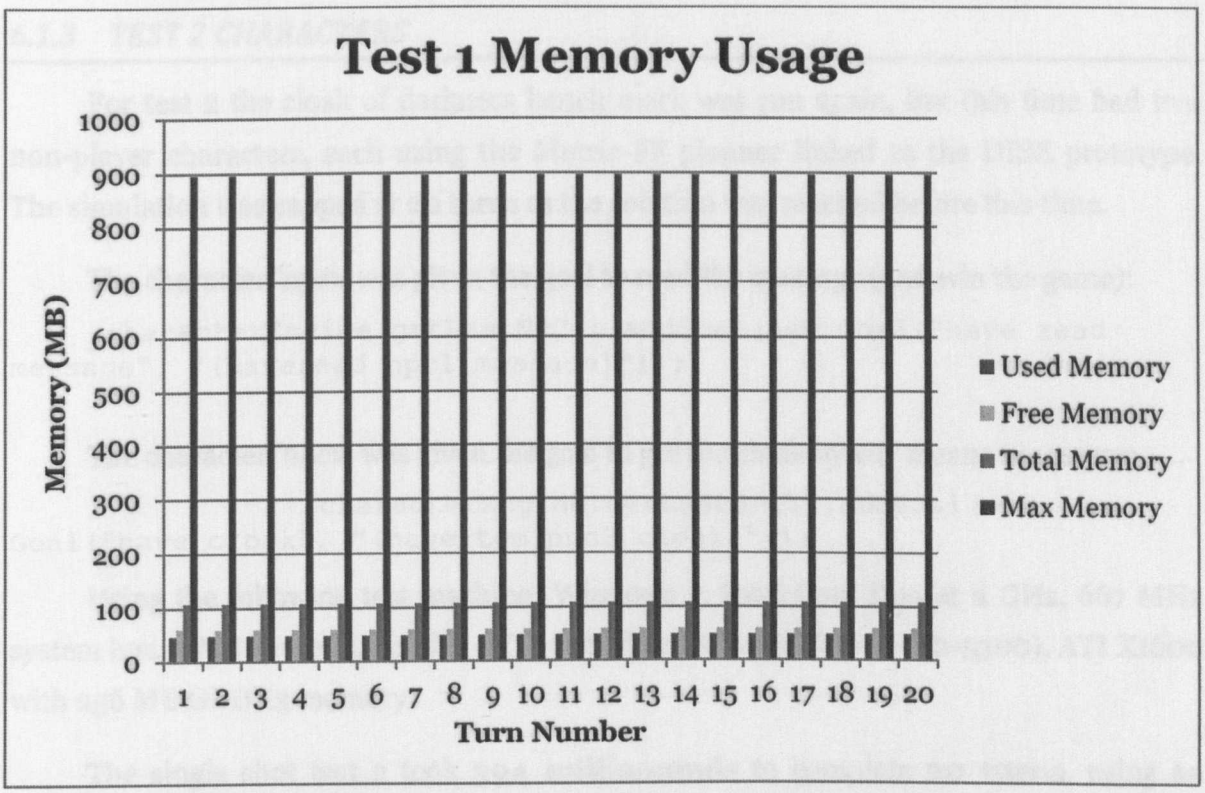


Figure 85: A Graph Showing the DISE CoD Test 1 Memory Usage

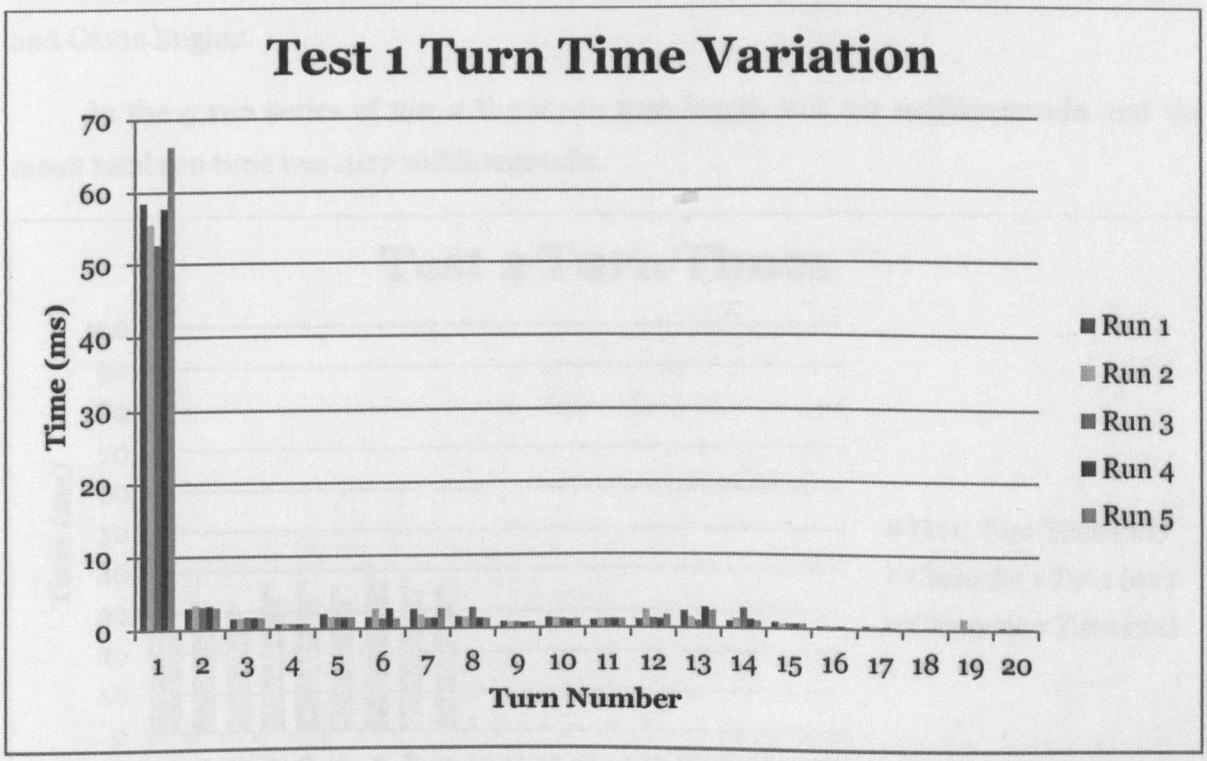


Figure 86: A graph to Show the Variation in Test 1 Turn Times over 5 Runs

For test 2 the cloak of darkness bench mark was run again, but this time had two non-player characters, each using the Metric-FF planner linked to the DISE prototype. The simulation was capped at 20 turns as the solution was reached before this time.

The character 'npc1' was given the goal to read the message (and win the game):

```
_characterEngine.getLastNPC().addGoal(new Goal("have read
message", "(haveread npc1 message)"));
```

The character 'npc2' was given the goal to get the cloak by any means necessary:

```
_characterEngine.getLastNPC().addGoal(new
Goal("have cloak", "(haveitem npc2 cloak)"));
```

Using the following test machine: Windows 7, Intel Core Duo at 2 GHz, 667 MHz system bus, 2 MB shared L2 cache, 2 GB 667 MHz DDR2 SDRAM (PC2-5300), ATI X1600 with 256 MB GDDR3 memory.

The single shot test 2 took **394 milliseconds** to complete **20 turns**, using an average of **46MB Memory**. The characters planning and thought process took **354 milliseconds** of this time, with the rest being used by the Story Manager, Action Engine and Game Engine.

In the 5 run series of test 2 the mean turn length was **20 milliseconds** and the mean total run time was **407 milliseconds.**

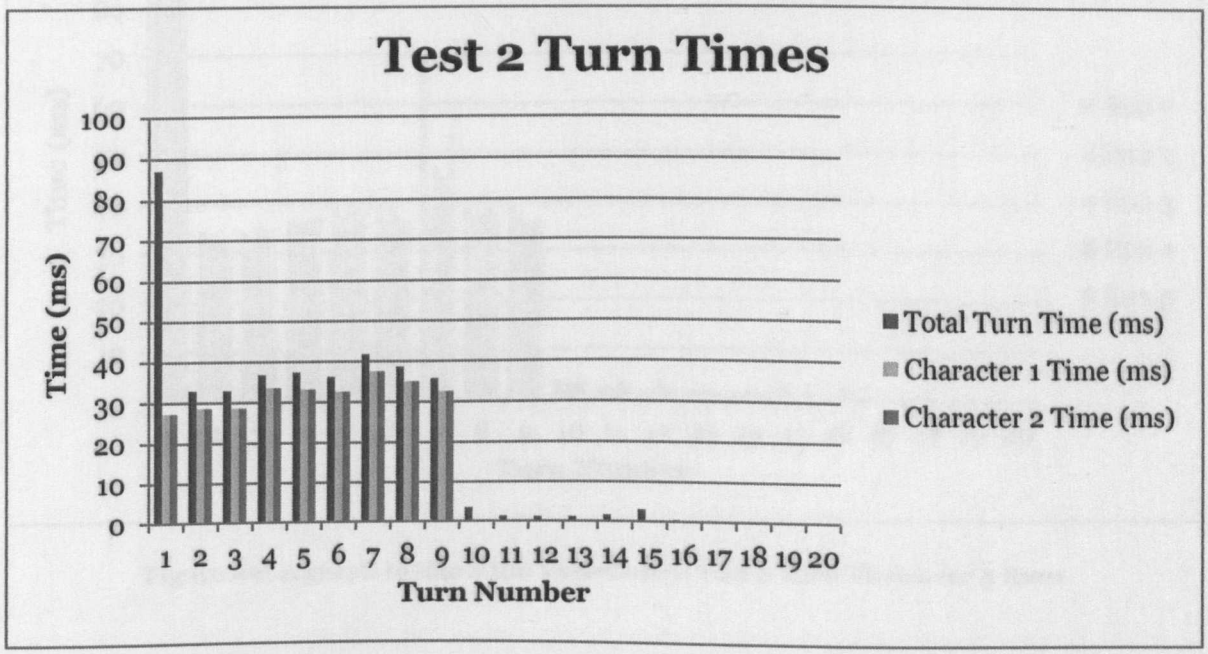


Figure 87: A Graph Showing the DISE CoD Test 2 Turn Times

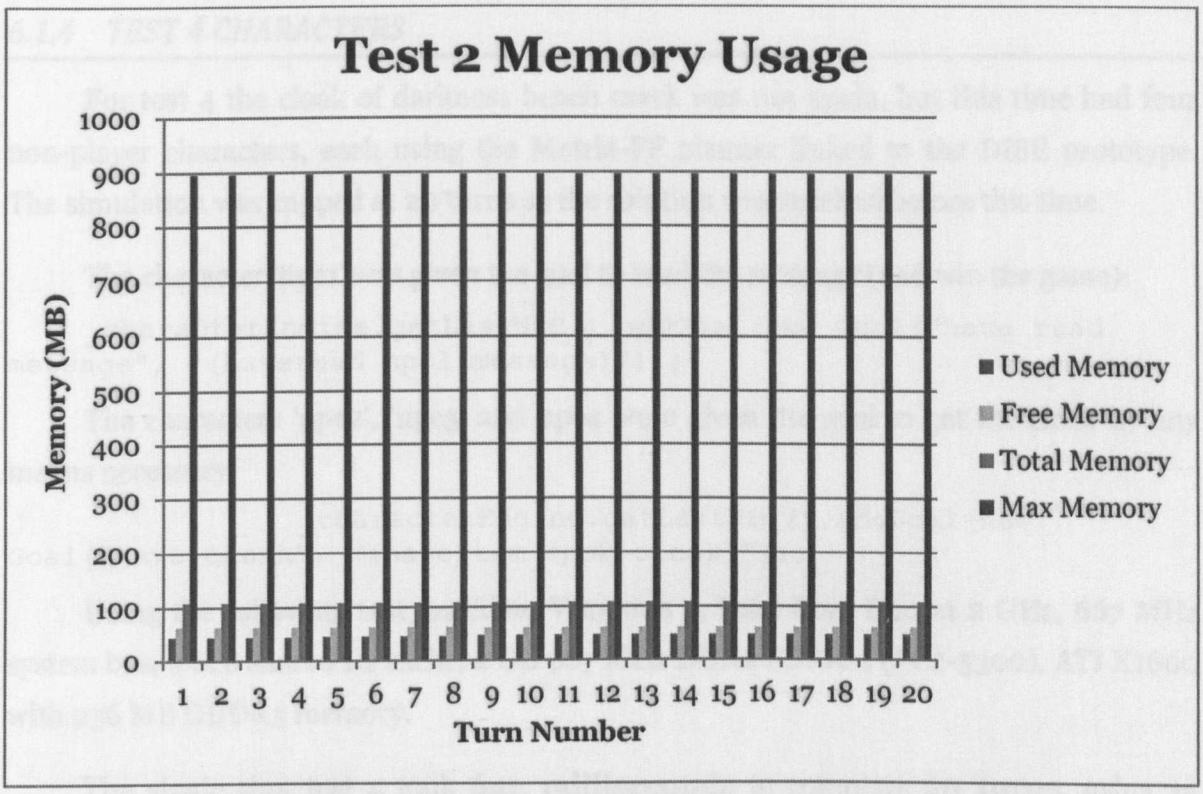


Figure 88: A Graph Showing the DISE CoD Test 2 Memory Usages

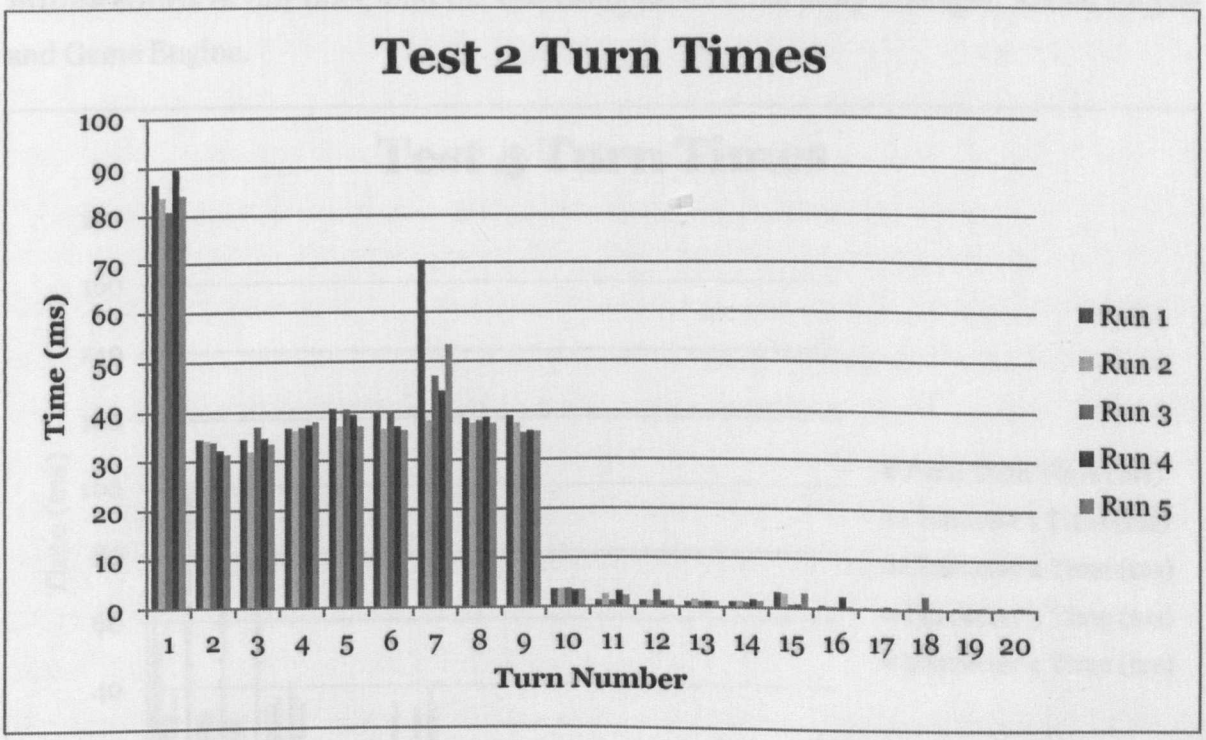


Figure 89: A graph to Show the Variation in Test 2 Turn Time over 5 Runs

For test 4 the cloak of darkness bench mark was run again, but this time had four non-player characters, each using the Metric-FF planner linked to the DISE prototype. The simulation was capped at 20 turns as the solution was reached before this time.

The character 'npc1' was given the goal to read the message (and win the game):

```
_characterEngine.getLastNPC().addGoal(new Goal("have read
message", "(haveread npc1 message)"));
```

The characters 'npc2', 'npc3' and npc4 were given the goal to get the cloak by any means necessary:

```
_characterEngine.getLastNPC().addGoal(new
Goal("have cloak", "(haveitem npc2 cloak)"));
```

Using the following test machine: Windows 7, Intel Core Duo at 2 GHz, 667 MHz system bus, 2 MB shared L2 cache, 2 GB 667 MHz DDR2 SDRAM (PC2-5300), ATI X1600 with 256 MB GDDR3 memory.

The single shot test 4 took **642 milliseconds** to complete **20 turns**, using an average of **68.7MB Memory**. The characters planning and thought process took **571 milliseconds** of this time, with the rest being used by the Story Manager, Action Engine and Game Engine.

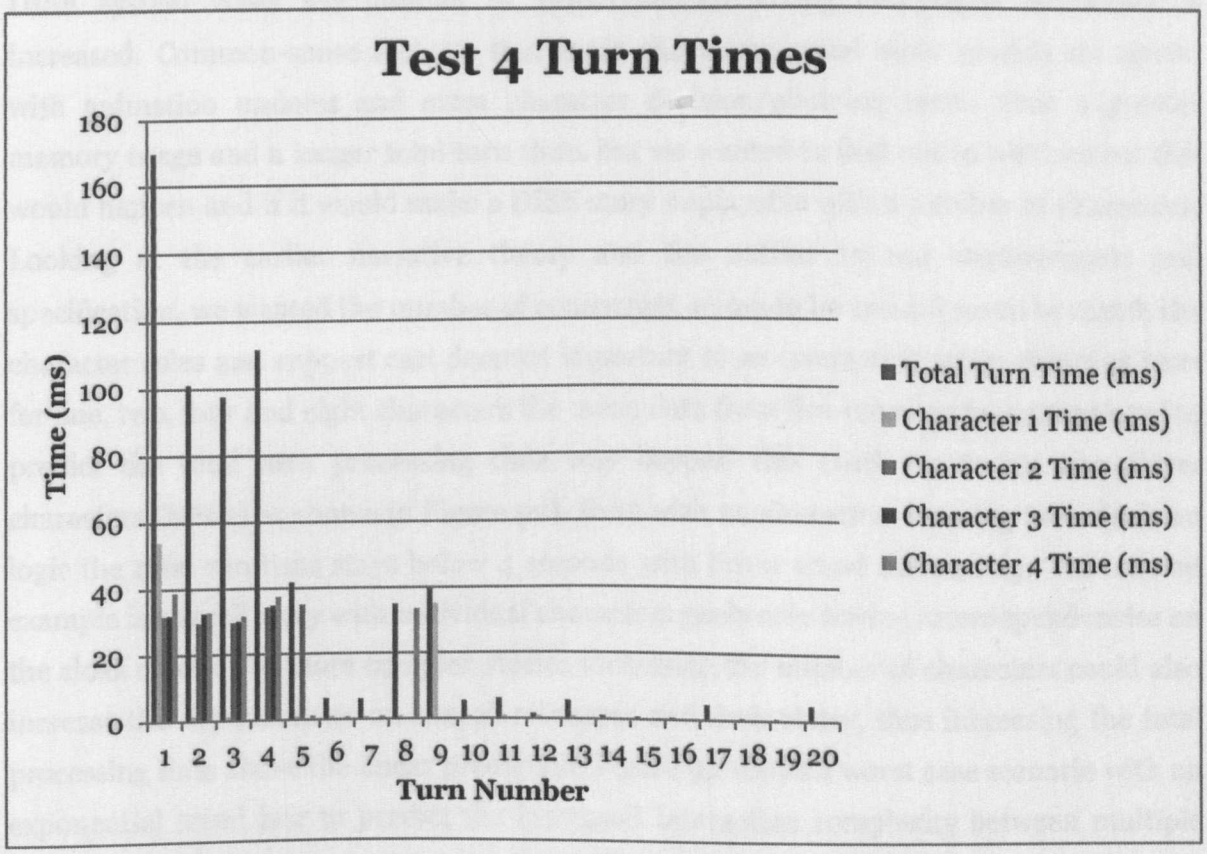


Figure 90: A Graph Showing the DISE CoD Test 4 Turn Times

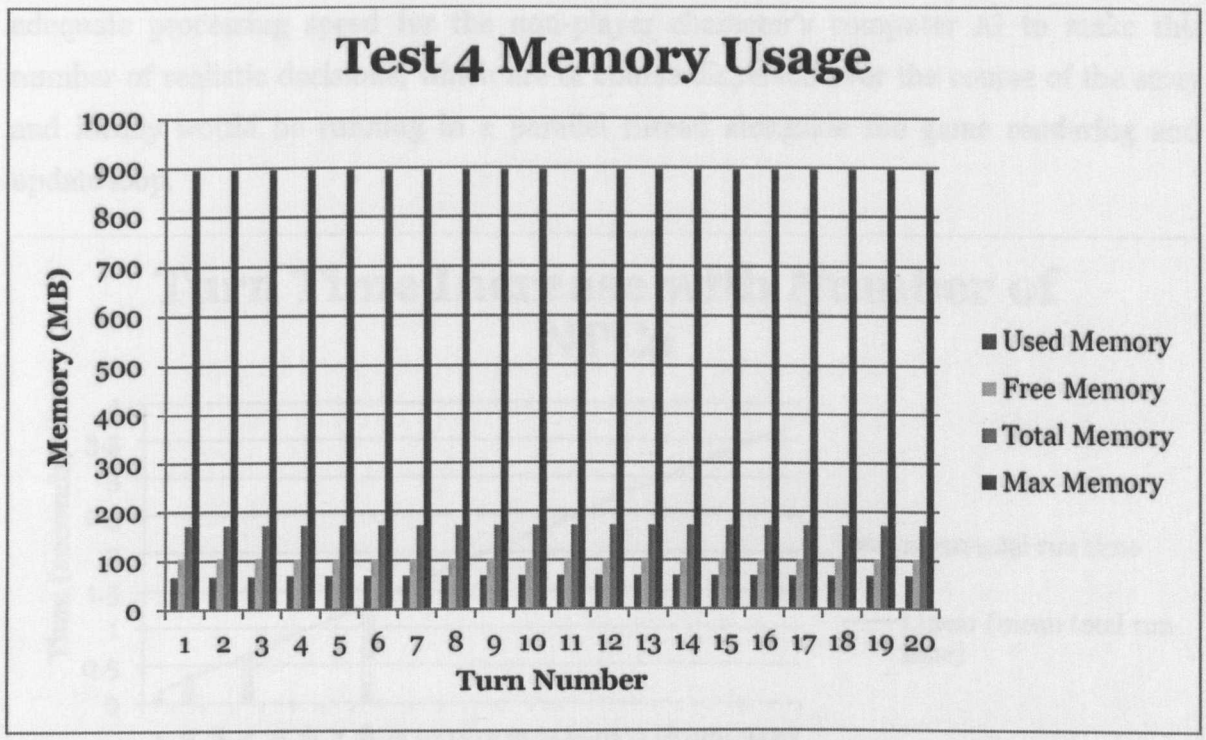


Figure 91: A Graph Showing the DISE CoD Test 4 Memory Usages

6.1.5 EXTRAPOLATION OF RESULTS

From our previous results the data can be used to predict the performance hit on the DISE system when the number of simultaneously acting non-player characters is increased. Common-sense dictates that more characters equal more models on screen with animation updates and extra character decision/planning turns, thus a greater memory usage and a longer total turn time, but we wanted to find out to what extent this would happen and if it would make a DISE story unplayable with x number of characters. Looking at the earlier narrative theory and the outline in our requirements and specification, we wanted the number of concurrent actors to be around seven to match the character roles and support cast deemed important to an interesting story. Running tests for one, two, four and eight characters the mean data from five runs can be extrapolated to predict the total turn processing time way beyond this (with up to 20 non-player characters (NPCs) as shown in Figure 92). Even with 20 characters running their decision logic the total run time stays below 4 seconds with linear trend forecasting. The current example is a small story with individual characters goals only having interdependencies on the cloak object. For more complex stories increasing the number of characters could also increase the dependencies on shared resources and their states, thus increasing the total processing time above the linear prediction. Figure 93 shows a worst case scenario with an exponential trend line to predict the increased interaction complexity between multiple characters with limited shared resources and goals that depend on them. This plots the total time at around 90 seconds to solve the Cloak of Darkness scenario with 20 characters. Even though this value is dramatically increased it would still provide

adequate processing speed for the non-player character's computer AI to make this number of realistic decisions, which are of course dispersed over the course of the story and ideally would be running in a parallel thread alongside the game rendering and update loop.

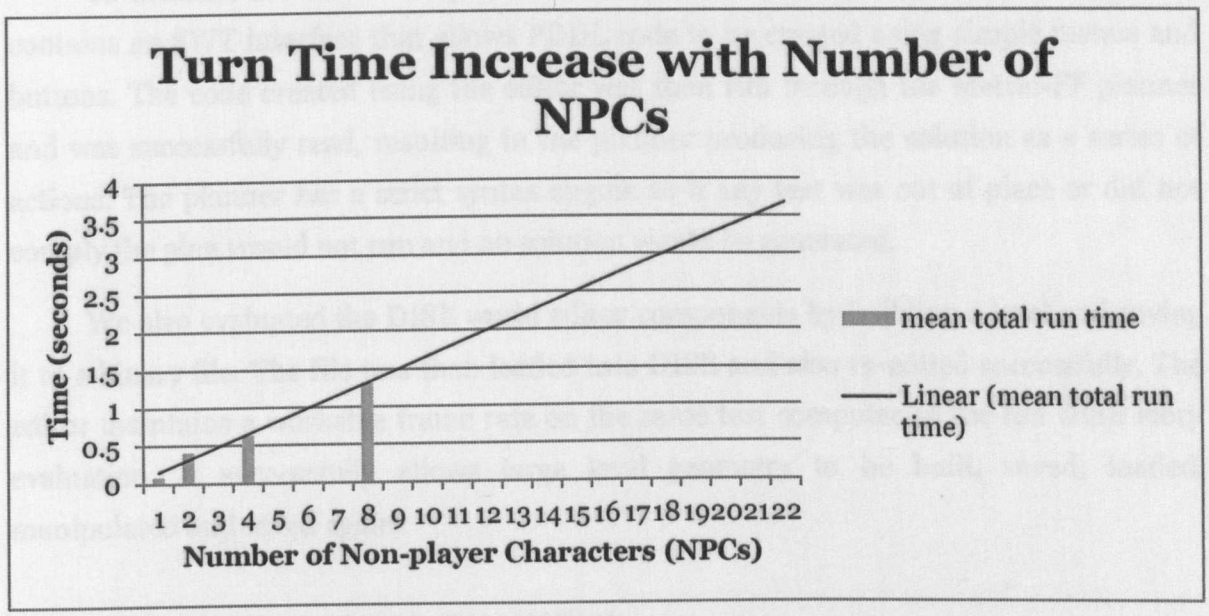


Figure 92: Linear Predicted Turn Time with Increased Character Numbers

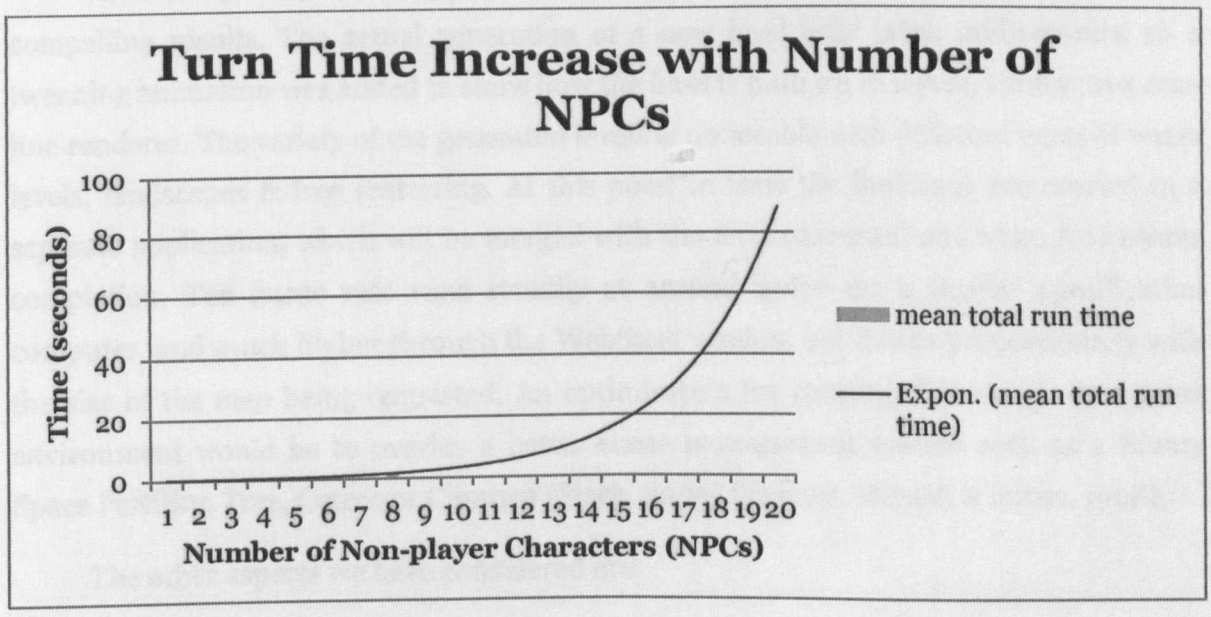


Figure 93: Exponential Predicted Turn Time with Increased Character Numbers

6.2 EVALUATION OF DISE EDITORS

6.2.1 DISE EDITOR EVALUATION

To evaluate the editor component we created another test application. This editor contains an SWT interface that allows PDDL code to be created using simple menus and buttons. The code created using the editor was then run through the Metric-FF planner and was successfully read, resulting in the planner producing the solution as a series of actions. The planner has a strict syntax engine so if any text was out of place or did not comply the plan would not run and no solution would be generated.

We also evaluated the DISE world editor components by building a level and saving it to a binary file. The file was then loaded into DISE and also re-edited successfully. The editor maintains a workable frame rate on the same test computer as the full DISE story evaluation. It successfully allows large level geometry to be built, saved, loaded, manipulated and saved again.

6.2.2 PROCEDURAL EDITOR EVALUATION

After completing a test application that runs a tile based map generator we had compelling results. The actual generation of a new level only takes milliseconds, so a tweening animation was added to show how the level is built up in layers, similar to a scan line renderer. The variety of the generated levels is noticeable with different types of water levels, landscapes & tree scattering. At this point in time the buildings are created in a separate application, which will be merged with the environmental one when it is nearer completion. The frame rate runs steadily at around 30fps on a regular specification computer, and much higher through the WebStart version, but deters proportionally with the size of the map being requested. An optimisation for running these maps in a game environment would be to overlay a better scene management system such as a Binary Space Partition Tree, Octree or Clipmap (Slack, 2009) (Tanner, Migdal, & Jones, 1998).

The other aspects we have considered are:

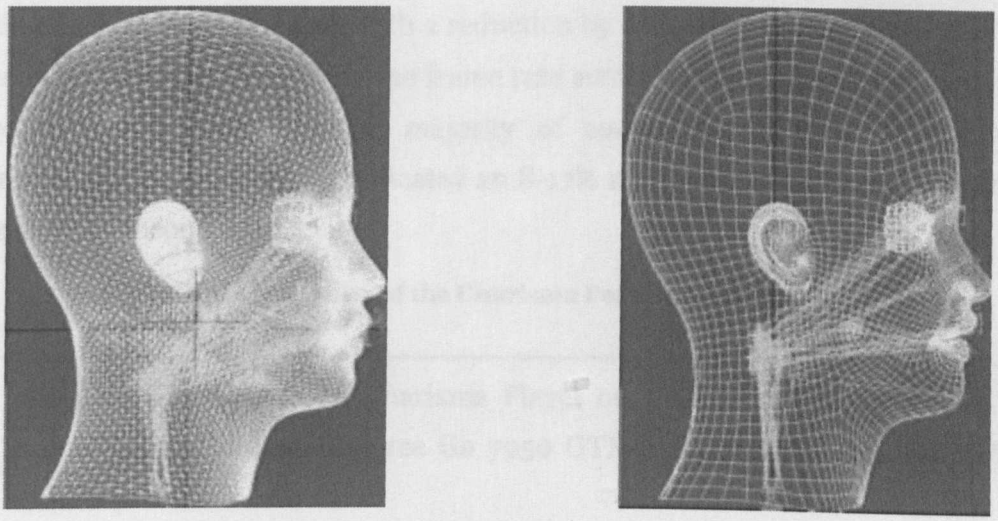
- Novelty: contains element of randomness and unpredictability.
- Structure: is not merely random noise, but contains larger structures.
- Interest: has a combination of randomness and structure that players find engaging.
- Speed: can be quickly generated.
- Controllability: can be generated according to a set of natural designer-centric parameters.

The program is still under development and a full usability evaluation will be

performed once it is completed, to analyse users' reaction to and satisfaction with the tools.

6.2.3 CHARISMA CHARACTER ANIMATION TOOL EVALUATION

In order to test the performance of Charisma framework, we used the Gabrielle model in combination with the Charisma player. The character is modelled as high-polygon (i.e. 62512 triangles) model (depicted in Fig. 94.a), medium-polygon model (i.e. 15560 triangles) and low-polygon (i.e. 8176 polygons) model (depicted in Fig. 94.b), with three light sources placed within the scene on elliptic orbits around the model, tested with a 540 frame FAP animation, and with a desired execution rate of 25FPS. The results are averaged over 20 executions of the tests and were performed on the Java Web Start variant of the applications. Typically, we expect an 8-10% performance drop when executed as an Applet embedded inside a web page (El Rhalibi, et al., 2009).



(a) High-polygon (62512) (b) Low-polygon (8176)

Figure 94: Models of Gabrielle used in the evaluations

In order to test the effect of the animation on the processing, we tested the application with full rendering of the model, but with no animation (animation loaded with updates disabled), the interpreted mode version and the compiled version. Three test types were performed: CPU-based skinning, where all transforms, lighting, texturing and bone manipulation are done on the CPU (typically used in low-end machines, without Shader Model 2.0 supporting GPUs); GPU-based skinning, where the aforementioned processes are carried out on the GPU and finally, flat shading, dynamic texturing and lighting is omitted – this is also a CPU bound process. Table 3 shows the results of the performance testing on the high-polygon model.

Table 3: Results of the Charisma Performance Test 1

Test 1: using the Homura-based Charisma Player on a Window 7-based PC with an Intel® Core™ 2 Duo E6600, NVidia 8500GT 256MB GPU, and 2GB DDR-800 System RAM.			
	Testing Mode (Results expressed as Frames Per Second)		
Test Type:	Static Model – No Animation	Interpreted Animation Mode	Compiled Animation Mode
CPU Skinned:	146.6	38.5	134.6
GPU Skinned:	85.1	36.7	74.5
Flat Shading:	268.3	40.4	222.5

The results across the animation techniques indicate that interpreted mode has a dramatic effect on the frame-rate with a reduction by a factor of between 2x-6x. This also produced in a levelling effect upon the frame rate across the test types, indicating that the interpreted update consumes the majority of computational time per frame. In comparison, the compiled mode indicated an 8-17% reduction in frame-rate, maintaining high frames per second output.

Table 4: Results of the Charisma Performance Test 2

Test 2: using the Homura-based Charisma Player on a Vista-based PC with an Intel® Core™ 2 Duo T7200, NVidia GeForce Go 7950 GTX 256MB GPU, and 2GB DDR-800 System RAM.		
	Testing Mode (Results expressed as Frames Per Second)	
Test Type: CPU Skinned	Static Model – No Animation	Compiled Animation Mode
62512 POLYGONS	131.4	117.9
15560 POLYGONS	361,7	283.3
8176 POLYGONS	589.6	334.2

Table 4 shows the results of the performance testing, and contrasting the high-polygon, medium-poly and low-polygon models using a slightly less powerful machine, for the static models and the compiled animation modes. Once again, the resulting frame rate achieved by the application exceeded the animation frame rate, resulting in very smooth animations.

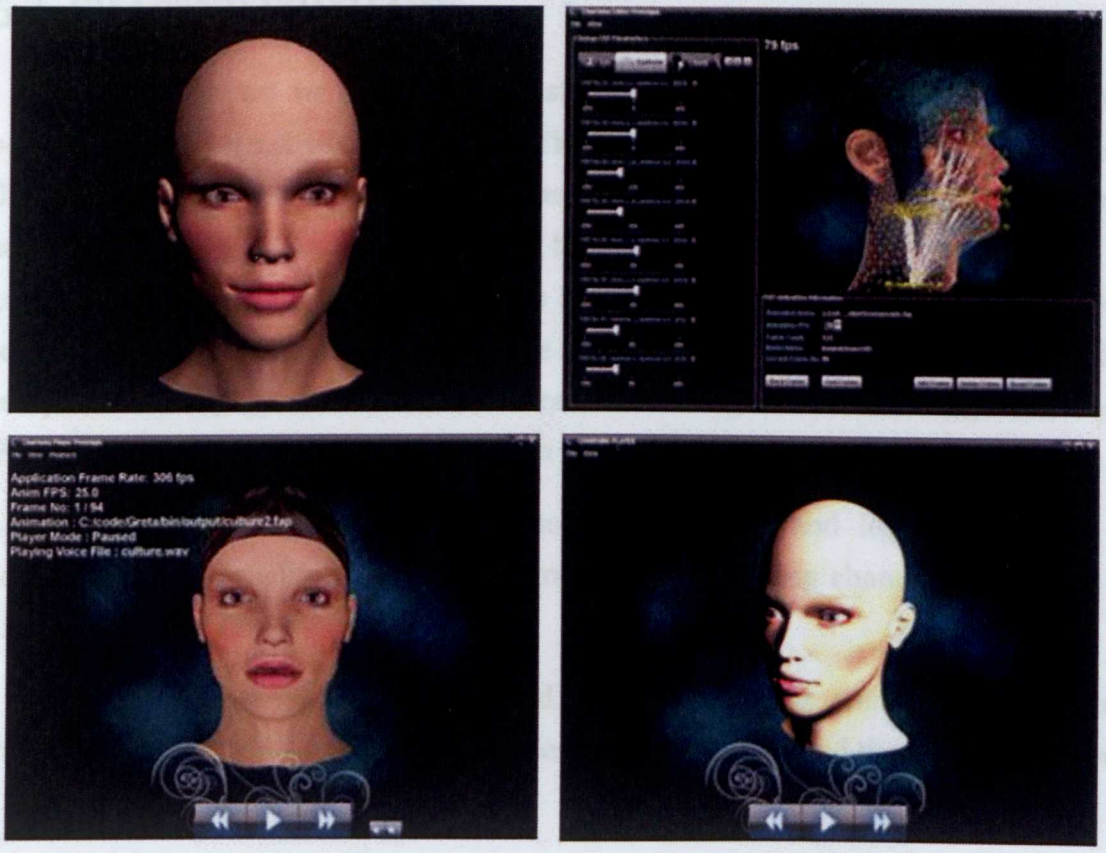


Figure 95: Charisma Prototype Application

The prototype applications shown in Figure 95 displays highly realistic models (top left) with the Charisma Editor (top right) and Charisma Player (lower two). The applications run as either standalone Java applications or JWS/Applets for web integration.

This chapter has proved the viability of the DISE framework by evaluating the test applications for both the storytelling engine and the content editing tools. The results show that characters can process their decisions and re-plan if the world changes around them, whilst still keeping memory usage and processing time to an acceptable amount (for example the average single turn time for one character is around 4 milliseconds). We can also see that the range of actions and level of customisability in the editors will allow a story world to be created that meets the requirements of a tested interactive fiction benchmark.

6.3 CHAPTER SUMMARY

In this chapter we evaluated our storytelling framework DISE using a popular benchmark for interactive fiction called "The Cloak of Darkness". This benchmark proves that DISE is feature rich enough to represent a complex story scenario and a range of actions. We also used this benchmark to test the Character Engine's capability to solve goals for multiple characters. The results showed that multiple characters can run and solve the Cloak of Darkness scenario while keeping memory usage consistent and with an acceptable solving time (even with 20 characters running their decision logic the total run time was shown to be below 4 seconds with linear trend forecasting).

We also evaluated the DISE Editors testing save and load consistency, level generation speed/quality and the frame rates whilst running character animation with different levels of detail.

The next chapter will summarise and conclude the thesis, state our findings, known limitations and list the scope for future research to further expand the DISE framework.

> *"Game authors should embrace interactivity rather than treating it as a problem that needs a solution. And don't just say that storytelling isn't possible because of non-linearity (Samyn, 2005).*

7 CONCLUSIONS & FUTURE WORK

Our goal was to design and evaluate a more complete Interactive Storytelling engine and framework/middleware, called 'The Digital Interactive Storytelling Engine' (DISE); which consists of separate player and editor components for the creation and deployment of new interactive story modules.

In this thesis we have described a system that allows an author to more easily create an interactive story with multiple outcomes that gives a greater level of freedom to the player. By specifying the places, objects, characters and their personalities along with the key scenes, sequences and the actions a player can choose from; a richer and more dynamic narrative can be constructed. The author has the ability to choose how much constraint they want to place on the player and what level of importance each section of the story has. Authors can also direct their stories to different age groups by removing inappropriate actions and content. These features along with real-time first person interaction in a 3d world, with the player being a main character and having a direct involvement in story progression, a context sensitive action menu, individual personalities and switchable planners for each character, and the ability to launch the game from a web-based 3D application using our game engine and deployment framework called 'Homura' (Carter et al., 2008), provide a novel take on Interactive Storytelling.

We identified the key design features and the most suitable narrative theories and computational models for use in our Interactive Storytelling framework. We used key parts of the following narrative theories in the design of the DISE framework: Aristotle's Dramatic Actions, Five-act Model, Three-act Paradigm, Roles & Processes, Narrative Grammars, Narrative Units, Five Codes of Analysis, and Ethical Dimension. We also implemented an editing system that allows the author to choose which narrative theory to recreate for themselves (for example a scene structure created with the scene editing tool allows the 3-act or 5-act model to be followed). The best fit computational model was a planning system, as goals and actions can be decoupled and created modularly, which allows a story to be greatly changed by adding a new action or goal in the editor.

Other novelties of our DISE framework include: the real-time facial animation system for storytelling with characters; the user friendly editors to manage story data,

describe characters, 3d word/level design/editing and procedurally creating art content; the pluggable planning system for further research and expandability; the development of integrated solution for DISE Framework and it's evaluation including the dissemination of our findings.

7.1 THESIS SUMMARY

In this thesis we have introduced the concept of Interactive Storytelling and described how it presents the opportunity for players to have an input on what is happening in the game world they are placed in, to be the ones who dictate how certain events may come to pass.

In Chapter one we outlined our goal, which was to design and evaluate a more complete Interactive Storytelling engine and framework/middleware, called 'The Digital Interactive Storytelling Engine' (DISE); which will consist of separate player and editor components for the creation and deployment of new story modules.

In Chapter two we presented the background of this research area to demonstrate a wider appreciation of the subject (to give context), and provide our problem statement and motivations for this thesis.

In Chapter three we surveyed and critically assessed the projects and publications related to the field of Interactive Storytelling, planning, and computer games technology and stated their relation to our own work, along with their positive and negative aspects. This included analysing the importance of narrative theories and computational models in the design of our Interactive Storytelling strategy and examining how a story can be formalised and deconstructed into its core components to systematically generate narrative elements.

In Chapter four we outlined the DISE framework and the game technologies we would need to implement it. We also looked at who will be using the framework and what level of interactions they will require.

In Chapter five we explained our DISE framework in detail using example flowcharts and some code examples to show how it was implemented and how both players and story authors can use DISE to respectively play and create new interactive story content. This implementation contained the main systems in DISE, including the Game Engine, Story Manager, Planner, Player Action Engine, Character Engine, Charisma Character Animation System and the editors to create all the necessary content including procedurally generating level mesh models.

In Chapter six we ran a fiction benchmark to assess if DISE was rich enough to

provide an Interactive Storytelling experience with a large number of simultaneous automated characters and a variety of goals and actions all in a real-time environment. We also looked at the performance of the character animation system and editing tools.

7.2 CONTRIBUTION

This section contains our hypothesis and identifies the problem to be explored and its importance to the field of Interactive Storytelling in Computer Games. It asserts that our research may help to solve the problem under investigation and is essentially a statement of what we believe the study will prove and/or solve including the novel aspects of our work and the contribution to knowledge.

We believe our study proves that the DISE framework provides a viable way to not only play a variety of real-time interactive stories, but to create new one by dedicating a small amount of time and creative input. We have provided an in depth review of other Interactive Storytelling systems created in the past, stating what pieces are important and what elements were missing. We also looked at narrative theories to find what elements can be used to create a viable narrative model for Interactive Storytelling, whilst giving the author the freedom to choose the discourse of their narrative themselves without limiting scene structure or genres. Our implementation proves that multiple characters can make quick decisions based on their goals and personalities and carryout actions until either their goals are met or they need to re-plan.

7.3 COMPARISON TO RELATED WORK

In this section we will compare our DISE framework to other researcher's implementations of an Interactive Storytelling system. After analysing multiple storytelling system implementations using various techniques, Crawford found that there are several core components and technologies that make up what is considered to be a comprehensive interactive storytelling engine (Crawford, Chris Crawford on Interactive Storytelling, 2004):

- **Story Actions and Events**
  - These are the actions that the user can do, multiple verbs make up events and are stored in flat data structures; an example from Crawford's engine is the trading sentence, "Subject Trades X (to) Direct Object (in return for) Y" (Crawford, Chris Crawford on Interactive Storytelling, 2004).
- **Drama and Story Managers**

- This is the main algorithmic game/story engine that links everything together. It updates the game world, the characters and their goals and evaluates events and their impact on the plot. It also remembers past events and learns from their results.

- **Personality and Emotional Models (Character Engine)**
  - These help the characters decide what decisions to make depending on certain character traits represented as dynamic numerical values that can change depending on interactions with other characters; for example an anger value may increase if insulted.
- **Roles and Sequencing Engine**
  - The sequencing engine's job is to calculate what should happen next after an event. It makes sure the options available to a player are relevant to the previous event and that characters choices fit their role correctly. Some of this will be moved into the Character Engine for our design.
- **History Database and Blackboard Systems**
  - History modules record events that characters do that could influence their future decisions and blackboard systems allow characters to share experiences and knowledge from their own point of view.
- **Anticipation Engine**
  - This system allows characters to anticipate another's reaction to their behaviour and adjust their actions accordingly.
- **Integrated Development Environment (IDE)**
  - These are the tools where story builders will specify, design, implement and evaluate their game stories. The IDE must be comprehensive and easy to use.
- **Game Engine**

The game engine handles all the logic behind the game and renders the graphics to the screen in an update loop.

DISE contains most of these components, but does not yet contain implementations of blackboard systems or anticipation engines.

Donikian & Portugal's Interactive Fiction system DraMachina (Donikian & Portugal, 2004), uses the architecture depicted below in Figure 96. The user interacts with a theatre to manipulate the story world model data and the author creates narrative logic which powers the interactive drama.

Figure 96: Architecture of an Interactive Fiction (Donikian & Portugal, 2004)

Our architecture uses most of the aforementioned components, but they are integrated in a slightly different way to create a more usable and interoperable structure, and support different technologies and data format. Our narrative logic exists as the story file and the user interacts via a contained class called the Player Action Engine. The theatre aspect could be represented by the game engine's rendering system.

We can use our contributions listed in section 1.4 to draw points of comparison to benchmark the features of DISE against five other competent Interactive Storytelling systems.

The main novel features we were aiming towards when designing and implementing our Interactive Storytelling system 'DISE' were to:

- Make a novel storytelling framework with full real-time interaction in a dynamic 3d world with the player taking the role of a main character in the story and looking through their eyes in a first-person perspective.
- Create useable editors to manage story data, describe characters and for 3d word/level design/editing and game assets.
- Create a switchable planning system for future extendibility.

So to compare these novel points we will look at the graphical output of the storyworld, the user's role, the systems extendibility and editors and also the target audience and genre, as shown in Table 5 below. The table cell is coloured green for features that are on a comparable level with our system 'DISE' and red for features that have been enhanced by or are novel to DISE.

Table 5: Comparing DISE to Other Interactive Storytelling Systems

FEATURES	DISE	FABULATOR	FAÇADE	FEARNOT	STORYTRON
GRAPHICS	3D	3D	3D	3D	2D Icons & Text
USER ROLE	The user explores freely and builds action sentences by clicking objects and choosing from available verbs and nouns.	The user can use the mouse to click objects & chooses actions to carry out from a list.	The user inputs natural language text to talk & uses the mouse and keyboard to move.	The user watches clips then advises on next action via text chat interface.	The user chooses options from a dropdown menu in response to story events.
EXTEND-ABILITY AND EDITORS	GUI editors for story and graphics generation and creation. Pluggable planner for expansion.	FABULATOR has source files for the planner which can be modified but no editors or source code distribution.	Commercial - Not designed to be modified so is a closed system with no authoring tools.	Source code would need heavy modification to extend e.g. to add the player as a character.	The Story World Authoring Tool "SWAT" can be used to create new storyworlds, but is complex.
AUDIENCE / GENRE	Computer Users / Any.	Computer Users / Mysteries & Who-done-it's.	Adults / Relationships.	School children & teachers / Bullying Education.	Computer Users / World leader simulation (but potentially any).
COMPUTAT-IONAL MODEL SUPPORT	Automated Planning & Scheduling	Automated Planning & Scheduling	Automated Planning & Scheduling	Automated Planning & Scheduling	Automated Planning & Scheduling

7.4 LIMITATIONS

Our DISE framework provides the tools and game technologies to create new interactive stories and for users to play through them. A main limitation is that no matter how many tools are provided; interactive stories still need time and dedication from the story author to create. This is due to the fact that every object, location, character and action needs to be created and given their appropriate values and the structure of the scenes and predicate logic needed to move between them still require a certain level of experience. In Interactive Storytelling there is not just a balance between story constraints and player freedom, but also between the complexities of story authorship/creation and the level of control and unique detail in the final story. If the author wants their story to be complex and flow a certain way, then high level authoring tools are not expressive enough. In the DISE framework the tools allow the construction of new predicates, actions, character personalities, 3d models and animations, animated character dialogues, goals and scene events and structures. Using these a custom unique story world can be created, but at the cost of more development effort on the authors behalf.

Other limitations that can be dealt with in the future and are mentioned in the next section 7.5 FUTURE WORK are:

- The feature constructs (such as durative actions) and solving speed of the planning system.
- The behavioural completeness of the characters' emotional model.
- The lack of history, gossip and deception features in DISE, to create more dynamic characters and therefore stories.
- The number of NPCs running decisions simultaneously in a single scene (including scenes that need large crowds).
- The number of polygons used in the close up animated character model scenes using Charisma could slow down the frame-rate when multiple characters are present.
- The world editor meshes could be stored in a more optimised graphical structure such as an Octree or in chunks, so that new areas are loaded faster, the environments can be larger and the polygon count of objects further away from the player are lowered to maintain an optimal frame-rate.
- Our main analysis of the DISE prototype applications at this stage provide quantitative data representing the soundness of the solution and characters decision processes, even with multiple characters, but we have not yet measured using an objective user analysis technique to see what users think of story and interface quality.

7.5 FUTURE WORK

This section contains information on missing functionality, descriptions of variations, extensions, or other applications of our central idea along with the possibilities for future research. Although we have provided many technical demos of certain aspects of the DISE system, there is still much more work that can be done to create the finished product, as stated earlier: "The goal [for research projects] is not to actually finish the software, but to demonstrate that it could be finished with an adequate expenditure of money and resources. Realistically one graduate student working for two years can't be expected to build a working system for interactive storytelling" (Crawford, Chris Crawford on Interactive Storytelling, 2004).

As the DISE Framework covers many areas and is composed of multiple systems and theories, we have divided up the future work that could be undertaken to enrich our framework into the following areas: planning, characters, editors and analysis.

7.5.1 PLANNING

Durative Actions – Although DISE characters can carry out an action over multiple turns in using the Character Engine to check if the current action is complete, it does not use durative actions in the planning stage. Durative actions (Fox & Long, 2003) give a specific duration to an action and categorise the time conditions are tested using three time groups: at the start of the action, at the end of the action and over all the whole action. The actions effect is also labelled as 'at start' (for immediate effects) or 'at end' (for delayed effects). This extra set of information allows concurrent planning (the use of resources by multiple actions). The example below shows the durative action for burning a match whilst simultaneously picking up a coin.

```
(:durative-action burn-match
:parameters (?m - match)(?l - location)
:duration (and
(< ?duration 5)(> ?duration 0)
)
:condition (and
(at start (have ?m))
(at start (at ?l))
)
:effect (and
            (when
                  (at start (dark ?l))
                  (and
                  (at start (not (dark ?l)))
                  (at start (light ?l))
                  )
            )
            (at start (not (have ?m)))
            (at start (burning ?m))
```

```
                (at end (not (burning ?m)))

                (when
                        (at start (dark ?l))
                        (and
                        (at end (not (light ?l)))
                        (at end (dark ?l))
                        )
                )
        )
)
)

(:action pickUp
:parameters (?l - location ?o - object)
:precondition (and
(at ?l)
(onFloor ?o ?l)
(light ?l)
)
:effect (and
(not (onFloor ?o ?l))
(have ?o)
)
)
```

This extra level of planning logical detail could allow more realistic solutions to be generated and remove the one action at a time limitation DISE characters currently have. This would allow for a more dynamic and realistic story with characters that are even more intelligent than the current implementation.

Optimising Planner – the DISE prototype allows researchers to change the planner element to use another one which could be newer or use a different heuristic. Further work could be done in creating a unique planning system designed specifically for Interactive Storytelling. Using different heuristics or relational node graph hierarchies and ignoring the elements that are not needed for story planning; a more streamlined planning system could be attached to DISE and directly be compared to the Metric-FF planner in the current implementation example.

Optimising Plan Dataflow – the speed of the planning stage can be increased when a final planning system is chosen, but will sacrifice the ability to easily switch out planning systems. Currently planning data is saved as two temporary text files (a problem and domain file) in the PDDL format. These two text files are input parameters of the planner's system command line, as the majority of existing planning systems require their input in this format. Optimisation would avoid the creation of these text files and instead hold their data in a string buffer or other suitable data structure and pass the information directly to the planner. The downside would be that each new planner would need to have their source code available and be edited to accept this new method of input on top of the planner interface class usually needed for a new DISE planner.

Personality Models – the current personality model in DISE is expandable and uses numerical values for each trait. DISE could be expanded in the future to support other personality models such as the widely used OCC Model by Ortony, Clore and Collins (Ortony, 2003). The OCC model has 22 emotional categories shown in Figure 97 below.

Figure 97: OCC Model of emotion (Ortony, 2003)

Characters process events in three phases to generate an emotional response:

- Categorisation – finding which emotional categories are affected by evaluating the event, object or action.
- Quantification – the intensity of the affected emotions.
- Mapping – mapping the resulting changes of the 22 emotional categories usually to a lower number of emotions to produce the character's expressions and behaviour.

This model requires a rich world model with a large fact database to assign values to objects and low level goals to characters, such as staying alive, with food helping to achieve this goal thus having a positive emotional affect.

History and Blackboard Memory System – would allow characters to retain information of past events, to give an extra level of detail. In the current system they will know that they do not trust another character, but they do not retain the reason why. The memory recall could influence their future decisions in more detail. A blackboard would allow the character to share the event from their point of view (see gossip system next).

Gossip System – this would allow characters to share information between

themselves and other characters via gossiping. A character's personality model could use a gossip value to store their likeliness to spread news and also use the relationship values to allow them to chat with friends about characters that they do not like.

Deception – Fairclough and Cunningham's paper (Fairclough & Cunningham, 2003) mentions an extension to their Interactive Story Engine to include a Character Deception system. This would allow the characters to deceive and "inform the player of events that did not happen in the world". These fabricated events should be designed to manipulate the player and evoke a certain reaction.

Level of Detail (LoD) for NPCs – to optimise the NPCs decision making times, allowing more characters to be running simultaneously in a single scene a level of detail system would be needed. Characters can be prioritised in order of importance to the story and also their proximity to the player's character. NPCs that are further away and not currently visible and are also not doing anything of large importance in the story can be updated less in the Story Managers decision turns loop and also don't need to be animated. There are many level of detail techniques used in computer games today (mainly for geometry optimisation), to improve the amount of processing power used on things that are not as relevant.

Level of Detail (LoD) for Charisma Character Animations – the animated characters in Charisma are made from detailed meshes, so when more characters are in a shot the processing power used by Charisma increases. A level of detail system would allow the meshes of characters to be collapsed down to a lower number of individual polygons, focusing on areas that are not visible or with less focal interest and a lower concentration of facial contortion needed for expressions. Work is currently being carried out by our team on an extension to the Charisma system to achieve this as described in the paper (Duarte, El Rhalibi, Merabti, Carter, & Cooper, 2011).

7.5.3 EDITORS

World Editor – the world editor prototype allows structures to be created but can be further developed to allow for a greater number of polygons and larger map sizes. The user creates and edits box objects to build their scene, so an optimisation would be to delete unseen inside faces where two boxes are adjacent and touching. The box class already contains the code to hide selected faces of a box using a Boolean setter function and the corresponding face's world axis aligned direction (North, East, South, West, Top and Bottom). When the level is saved an algorithm could be created to iterate over each box to check if any hidden faces can be removed.

Another future improvement would be the automatic generation of navmesh data in

the editors. Although it is mentioned in this thesis for the design of DISE the current demo implementation doesn't include this feature yet due to time restraints. The navmesh would include any faces considered to be walkable floor space with enough clearance above it for a character to walk on standing up. The navmesh can be pre-generated after the level design is saved and could also write the accessibility data into the planning data files as mentioned in section 5.4.13.

7.5.4 IMPROVING ANALYSIS

The DISE system was measured for its performance, diversity and functionality and results were also projected to test the number of simultaneous characters and the output solution from DISE listing the characters' actions can be found in APPENDIX 1 – Test 1 Console Output. Fairclough and Cunningham suggest that further work could be carried out to gain an objective analysis of the story from their users "In the interest of a **more objective analysis**, the networked storytelling system will be combined with a separate user interface for getting feedback on the users' experiences with the game, based on story criticism criteria. The result of this will include an analysis of believability, consistency, drama, and the level of user interactivity" (Fairclough & Cunningham, 2003).

For the evaluation of the Defacto system (section 3.1.6), Sgouros used user feedback questionnaires to evaluate the clarity of player's goals and the relevancy of their actions, along with the overall quality of the interactive story experience. Some of the questions also evaluated the clarity of the user interface. In the future when the DISE prototype applications are extended this method of evaluation would be useful for assessing the quality of the framework including both the playing and editing functionality.

BIBLIOGRAPHY

Allbeck, J. M. (2010). CAROSA: A Tool for Authoring NPCs. In R. Boulic (Ed.), *MIG 2010* (pp. 182-193). Utrecht, The Netherlands: Springer.

Aristotle. (1961). *Poetics*. (F. Fergusson, Ed., & S. H. Butcher, Trans.) Retrieved September 21, 2011, from The Internet Classics Archive: http://classics.mit.edu/Aristotle/poetics.html

Axelrad, M., & Szilas, N. (2010, Feburary). *IRIS Wiki - IS Systems*. (IRIS) Retrieved September 21, 2011, from IRIS Wiki: http://iris.scm.tees.ac.uk/public-wiki/index.php/IS_Systems

Bae, B.-C., & Young, R. M. (2008). A Use of Flashback and Foreshadowing for Surprise Arousal in Narrative Using a Plan-Based Approach. In Spierling, & Szilas (Ed.), *Interactive Storytelling - ICIDS 2008* (pp. 156-157). Erfurt, Germany: Springer.

Barber, H. (2008, January). *Interactive narrative*. (University of York, Department of Computer Science) Retrieved September 21, 2011, from University of York, Department of Computer Science: http://www-users.cs.york.ac.uk/~maria/gied/

Barber, H., & Kudenko, D. (2007). Dynamic Generation of Dilemma-based Interactive Narratives. *The Third Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE 07)*. Stanford, CA: Stanford University.

Barber, H., & Kudenko, D. (2008). Generation of Dilemma-based Interactive Narratives with a Changeable Story Goal. *International Conference on Intelligent Technologies for interactive entertainment.* Playa del Carmen, Mexico.

Barros, L. M., & Musse, S. R. (2005). Introducing Narrative Principles into Planning-based Interactive Storytelling. *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology. 265.* Valencia, Spain: ACM.

Barros, L., & Musse, S. (2007). Planning Algorithms for Interactive Storytelling. *Computers in Entertainment (CIE) - Interactive entertainment. Volume 5 Issue 1 Article 4.* New York, NY, USA: ACM.

Barros, M., & Musse, R. (2007). Improving Narrative Consistency in Planning-Based Interactive Storytelling. *Proceedings of The Third Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE 07).* Stanford, CA: Stanford University.

Bassos, M. (2004, September 24). *Memorable Video Game Characters.* (PAL Gaming Network) Retrieved September 21, 2011, from PALGN: http://palgn.com.au/article.php?id=1586

BBC News. (2009, January 10). *Games will 'eclipse' other media.* (BBC) Retrieved September 21, 2011, from BBC News: http://news.bbc.co.uk/1/hi/technology/7821612.stm

Berlyn, M., & Blank, M. (n.d.). *Interactive Fiction and the Future of the Novel.* Retrieved September 21, 2011, from Atari Archives: http://www.atariarchives.org/deli/interactive_fiction.php

Bickham, J. M. (1993). *Scene & Structure.* Writer's Digest Books.

Bicknell, S. (n.d.). *Five Coolest Mute Video Game Characters.* Retrieved September 21, 2011, from Shift-1: http://www.shift-1.net/mutes.html

BioWare. (2011). *Mass Effect.* Retrieved September 21, 2011, from http://masseffect.bioware.com/

Blum, A. (2001). *Graphplan HomePage.* (School of Computer Science, Carnegie Mellon University, Pittsburgh) Retrieved September 21, 2011, from School of Computer Science, Carnegie Mellon University, Pittsburgh: http://www.cs.cmu.edu/~avrim/graphplan.html

Bremond, C. (1966). *The Logic of Narrative Possibilities.* (E. Cancalon, Trans.) New Literary History, The University of Virginia.

Cai, Y., Miao, C., Tan, A., & Shen, Z. (2006). Fuzzy cognitive goal net for interactive storytelling plot design. *Proceedings of the 2006 ACM SIGCHI international Conference on Advances in Computer Entertainment Technology. (Hollywood, California, June 14 - 16, 2006). 266.* New York: ACM.

Campbell, J. (1949). *The Hero with a Thousand Faces* (1st ed.). US: Pantheon Books.

Carter, C., Cooper, S., Dennett, C., & Sabri, H. (2008). *Web Starts.* (JMU) Retrieved 2011 йил 21-September from Homura Game Development IDE: http://java.cms.livjm.ac.uk/homura/downloads.php

Carter, C., Cooper, S., El Rhalibi, A., Merabti, M., & Price, M. (2010). The Application of an MPEG-4 Compliant Animation to a Modern Games Engine and Animation Framework. *Lecture Notes in Computer Science 2010, 6459/2010,* pp. 326-338.

Cavazza, M., Charles, F., & Mead, S. (2002 йил May). Planning Characters' Behaviour in Interactive Storytelling. *The Journal of Visualization and Computer Animation, 13*(2), 121-131(11).

Cavazza, M., Charles, F., & Mead, S. J. (2001). Narrative Representations and Causality in Character-Based Interactive Storytelling. *CAST 2001* (pp. 139-142). Teesside: University of Teesside, UK.

Cavazza, M., Charles, F., & Mead, S. J. (2002). *Sex, Lies, and Video Games: an Interactive Storytelling Prototype.* Middlesbrough: University of Teesside CMS.

Cavazza, M., Pizzi, D., Charles, F., Vogt, T., & Andre, E. (2009). Emotional Input for Character-based Interactive Storytelling. *The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, (pp. 313-320). Budapest, Hungary.

Champagnat, R., Estraillier, P., & Prigent, A. (2006). Adaptative execution of game: unfolding a correct story. *Proceedings of the 2006 ACM SIGCHI international Conference on Advances in Computer Entertainment Technology (Hollywood, California, June 14 - 16, 2006). 266.* New York: ACM.

Christopher. (2010, March 18). *Writing a Screenplay with the Syd Field 3-Act Paradigm.* Retrieved September 21, 2011, from scriptxray.com: http://www.scriptxray.com/writing-a-screenplay-with-the-syd-field-3-act-paradigm/

Ciarlini, A., Pozzer, C., Furtado, A., & Feijó, B. (2005). A logic-based tool for interactive generation and dramatization of stories. Valencia, Spain : ACM.

Compton, J. (2010). *Character Point of View and CRPGS.* (Planewalker Games LLC) Retrieved September 21, 2011, from Planewalker Games: http://www.planewalkergames.com/broken-hourglass/developer-articles-mainmenu-57/103-pov

Cooper, S., El Rhailbi, A., Merabti, M., & Price, M. (2010). DISE: The Digital Interactive Storytelling Engine. *Edutainment 2010.* Changchun, China.

Cooper, S., El Rhalibi, A., & Merabti, M. (2011). DISE: A Game Technology-based Interactive Storytelling Framework. *The 12th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNET 2011).* Liverpool.

Cooper, S., El Rhalibi, A., Merabti, M., & Price, M. (2008). Dynamic Interactive Storytelling for Computer Games Using AI Techniques. *6th International Conference in Computer Game Design and Technology (GDTW).* Liverpool: LJMU.

Cooper, S., El Rhalibi, A., Merabti, M., & Price, M. (2010). DISE: The Digital Interactive Storytelling Engine. *The 11th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Braodcasting (PGNET 2010).* Liverpool.

Cooper, S., El Rhalibi, A., Merabti, M., & Wetherall, J. (2010). Procedural Content Generation and Level Design for Computer Games. *AISB 2010.* Leicester.

Costikyan, G. (2005). Constraining Interaction to Create Emergent Narrative. Tampere: University of Tampere.

Crawford, C. (2004). *Chris Crawford on Interactive Storytelling.* Berkeley: New Riders.

Crawford, C. (2004). Personality Modelling for Interactive Storytelling. *IE2004 Australian Workshop on Interactive Entertainment.* Sydney: University of Technology Sydney.

Crawford, C., & Mixon, L. J. (2008). *Storytron Interactive Storytelling.* (Storytron, Inc) Retrieved September 21, 2011, from http://www.storytron.com/: http://www.storytron.com/

De Sevin, E., & Thalmann, D. (2005). A Motivational Model of Action Selection for Virtual Humans. *Computer Graphics International (CGI)* (pp. 213-220). New York: IEEE Computer Society Press.

Dena, C., Douglass, J., & Marino, M. (2005). Benchmark Fiction: A Framework for Comparative New Media Studies. *Digital Arts and Culture Conference,* (pp. 89-98). Bergen, Norway.

Dennett, C., El Rhalibi, A., Fergus, P., Merabti, M., Cooper, S., Sabri, M. A., et al. (2008). 3D Java Game Development with Homura. *6th International Conference GDTW 2008*. Liverpool.

Dirks, T. (2010). *Film Terms Glossary*. Retrieved September 21, 2011, from http://www.filmsite.org/filmterms15.html

Donikian, S., & Portugal, J. (2004). Writing Interactive Fiction Scenarii with DraMachina. *Technologies for Interactive Digital Storytelling and Entertainment, Lecture Notes in Computer Science, 3105*, 101-112.

Duarte, R., El Rhalibi, A., Merabti, M., Carter, C., & Cooper, S. (2011). An MPEG-4 Compliant Quadric-Based Surface Adaptative LOD. *The 12th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNET 2011)*. Liverpool, UK.

ecirweb. (2006, April 26). *The FearNot! demonstrator*. Retrieved January 22, 2011, from ecircus: http://www.macs.hw.ac.uk/EcircusWeb/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=13&MMN_position=37:37

El Rhalibi, A., Baker, N., & Merabti, M. (2005). Emotional agent model and architecture for NPCs group control and interaction to facilitate leadership roles in computer entertainment. *Advances in Computer Entertainment Technology*, (pp. 156-163).

El Rhalibi, A., Dennett, C., Merabti, M., Fergus, P., Cooper, S., Ariff Sabri, M., et al. (2008). Homura: A Step Further Toward 3D Java Game Development Support. *ACM ACE 2008*.

El Rhalibi, A., Dennett, C., Merabti, M., Fergus, P., Cooper, S., Ariff Sabri, M., et al. (2009). 3D Java Web-Based Games Development and Deployment. *IEEE International Conference on Multimedia Computing and Systems 2009, Volume: 2, Issue: 3-4*, pp. 553 - 559. Ouarzazate, Morocco.

El Rhalibi, A., Merabti, M., Carter, C., Dennett, C., Cooper, S., Sabri, M. A., et al. (2009). 3D Java Web-Based Games Development and Deployment. *International Journal on Information and Communication Technologies, 202.*

El Rhalibi, A., Merabti, M., Price, M., & Cooper, S. (2008). Homura Platform: Integrated Spatial Editor and IDE for Games Development. *HCI 2008.*

Elrod, C. (2007). *Games and Storytelling - A Working Definition of Storytelling That Encompasses New Media.* Retrieved February 10, 2009, from PJ's Attic: http://www.pjsattic.com/?dl=1

Fairclough, C. (2004). *Story Mechanics as Game Mechanics: Applying Story Analysis Techniques to Game Artificial Intelligence.* Retrieved September 21, 2011, from ercim.org: http://www.ercim.org/publication/Ercim_News/enw57/fairclough.html

Fairclough, C., & Cunningham, P. (2003). A Multiplayer Case Based Story Engine. Dublin, Ireland.: Trinity College Dublin, Department of Computer Science.

Field, S. (1979). *Screenplay.* Delacorte Press.

Firth, R. (2010). *Cloak of Darkness.* Retrieved September 21, 2011, from Roger Firth's IF pages: http://www.firthworks.com/roger/cloak/

Forchheimer, R., Pandzic, I., & Pakstas, A. (2002). *MPEG-4 Facial Animation: The Standard, Implementation and Applications.* New York, USA: Wiley and Sons Inc.

Fox, M., & Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research, 20,* 61-124.

Freytag, G. (1900). *Freytag's Technique of the drama : an exposition of dramatic composition and art* (6th ed.). (E. J. MacEwan, Trans.) Chicago: Scott, Foresman.

Genette, G. (1983). *Narrative Discourse: An Essay in Method* (Reprint ed.). (J. E. Lewin, Trans.) Cornell University Press.

Glass, K. (n.d.). *Path Finding on Tile based Maps.* (K. Glass, Editor) Retrieved September 21, 2011, from Coke & Code: http://www.cokeandcode.com/index.html?page=tutorials/tilemap2

Haslum, P. (2003). *Writing Planning Domains and Problems in PDDL.* (Australian National University) Retrieved September 21, 2011, from Australian National University: http://users.cecs.anu.edu.au/~patrik/pddlman/writing.html

Huber, M. J. (2001). *Intelligent Reasoning Systems.* (Executive Media Online) Retrieved September 21, 2011, from http://www.marcush.net/IRS/irs_downloads.html

IGDA. (2001). *Foundations of Interactive Storytelling.* Retrieved September 21, 2011, from IGDA.org: http://archives.igda.org/writing/InteractiveStorytelling.htm

Jimenez, S. (2010, November 10). *Competition Rules.* Retrieved June 25, 2011, from IPC Conference 2011: http://ipc.icaps-conference.org/

Kautz, H. A., & Selman, B. (2006). *Satplan.* Retrieved March 6, 2011, from http://www.cs.rochester.edu/~kautz/satplan/index.htm

Kearney, R. (2002). *On Stories* (1st ed.). London: Routledge.

Knight, T. (2002, March). *Architectural Design Workshops: Computational Design for Housing.* Retrieved September 21, 2011, from MIT Open Courseware: http://ocw.mit.edu/courses/architecture/4-184-architectural-design-workshops-computational-design-for-housing-spring-2002/lecture-notes/

Lester, P. (2005, July 18). *A* Pathfinding for Beginners*. Retrieved September 21, 2011, from gamedev.net: http://archive.gamedev.net/archive/reference/programming/features/astar/index.html

Lintfordpickle. (2009, August 23). *Procedural Road Generation*. Retrieved September 21, 2011, from Britonia Game Blog: http://britonia.wordpress.com/2009/08/23/procedural-road-generation/

Liu, D. (2004 йил 10-March). *Heuristic Search Planners*. (Artform of University of Huddersfield) Retrieved 2011 йил 21-September from University of Huddersfield: http://scom.hud.ac.uk/planet/repository/heuristic.html

Louchart, S., Aylett, R., Dias, J., & Paiva, A. (2006). Unscripted Narrative for affectively driven characters. *IEEE Journal of Graphics and Animation, 26*(3), 42-52.

Lozano, M., Mead, S., Cavazza, M., & Charles, F. (2002). Search-based Planning for Character Animation. *ADCOG 2002*, (pp. 41-43). Hong Kong, China.

Lucas Online. (2009). *The Secret of Monkey Island: Special Edition*. (Lucas Online) Retrieved September 21, 2011, from LucasArts.com: http://www.lucasarts.com/games/monkeyisland/

Lugrin, J., & Cavazza, M. (2006). AI-Based World Behavior for Emergent Narratives. *Proceedings of the ACM Advances in Computer Entertainment Technology*. Los Angeles, USA.

Maher, J. (2006). *Let's Tell a Story Together*. Retrieved September 21, 2011, from http://maher.filfre.net/if-book/index.html

Mallan, K. (2003). *Performing Bodies: Narrative, Representation, and Childrens Storytelling*. Flaxton, Qld.

Mateas, M., & Stern, A. (2003). Façade: An Experiment in Building a Fully-Realized Interactive Drama. *Game Developers Conference (GDC'03).* Atlanta: Literature, Communication and Culture and College of Computing, Georgia Tech.

Merrill, D., & Kalanithi, J. (2008). *Research and Coursework - Sifteos.* (MIT) Retrieved September 21, 2011, from MIT Media Lab: https://www.sifteo.com/

Murray, J. (2004). *First Person - From Game Story to Cyber Drama.* (N. Wardrip-Fruin, & P. Harrigan, Eds.) Retrieved September 21, 2011, from http://www.electronicbookreview.com/thread/firstperson/autodramatic

Nelmes, J. (Ed.). (2003). *An Introduction to Film Studies* (3rd Revised ed.). London: Routledge.

Newell, G. (2008, December 24). *Gabe Newell Writes for Edge.* (Future Publishing) Retrieved September 21, 2011, from edge-online.com: http://www.next-gen.biz/opinion/gabe-newell-writes-edge

Noyle, J. (2006). *Techniques of Written Storytelling Applied to Game Design.* Retrieved September 21, 2011, from Gamasutra.com: http://www.gamasutra.com/features/20060426/noyle_01.shtml

Ong, T., & Leggett, J. (2004). A Genetic Algorithm Approach to Interactive Narrative Generation. *HYPERTEXT '04 Proceedings of the fifteenth ACM conference on Hypertext and hypermedia* (pp. 181-182). New York, NY, USA: ACM.

OpenTTD Contribs. (2011, March 8th). *AI Road Pathfinder.* Retrieved September 21, 2011, from Open TTD Wiki: http://wiki.openttd.org/AI:RoadPathfinder

ORIAS. (2000). *Monomyth Website.* (University of California, Berkeley) Retrieved September 21, 2011, from The Office of Resources for International and Area Studies : http://orias.berkeley.edu/hero/

Orkin, J. (2006). Three States and a Plan: The A.I. of F.E.A.R. *Proceedings of the Game Developer's Conference (GDC)*. San Jose.

Ortony, A. (2003). On Making Believable Emotional Agents Believable. In R. Trappl, P. Petta, & S. Payr (Ed.), *Emotions in Humans and Artifacts* (pp. 189-212). London, England: MIT Press.

O'Sullivan, T., Hartley, J., Saunders, D., Montgomery, M., & Fiske, J. (1994). *Key Concepts in Communication and Cultural Studies (Studies in Culture and Communication)* (2nd Revised ed.). London: Routledge.

Pakinkis, T. (2011, February 24). *First-person viewpoint 'is most immersive' - Levine*. Retrieved April 20, 2011, from http://www.computerandvideogames.com/290598/news/first-person-viewpoint-is-most-immersive-levine/

Penguin Books Ltd. (2009). *We Tell Stories*. (Penguin Books Ltd) Retrieved September 21, 2011, from We Tell Stories: http://wetellstories.co.uk/

Perlin, K. (n.d.). *A sheet of simplex noise*. Retrieved 09 21, 2011, from Ken Perlin's Homepage: http://mrl.nyu.edu/~perlin/homepage2006/simplex_noise/index.html

Pizzi, D., & Cavazza, M. (2007). Affective Storytelling based on Characters' Feelings. *AAAI Fall Symposium on Intelligent Narrative Technologies*. Arlington, Virginia.

Powell, M. (n.d.). *Class SharedMesh*. Retrieved January 10, 2010, from JME Java Docs: http://www.jmonkeyengine.com/doc/com/jme/scene/SharedMesh.html

Prigent, R., Champagnat, R., & Estraillier, P. (2005). Driving stories, benefits of properties analysis.

Princeton University. (2011, February 3). *WordNet - A Lexical Database for English*. (Princeton University) Retrieved January 20, 2011, from Princeton University: http://wordnet.princeton.edu/

Propp, V. (1977). *Morphology of the Folktale.* University of Texas Press.

Quantic Dream. (2011). *Heavy Rain.* Retrieved March 22, 2011, from http://www.heavyrainps3.com/

Rank, S. (2004). *ActAffAct.* Retrieved February 10, 2011, from OFAI Austrian Research Institute for Artificial Intelligence: http://www.ofai.at/research/agents/projects/actaffact.html

Rank, S. (2004). *Affective Acting: An Appraisal-based Architecture for Agents as Actors.* Retrieved September 21, 2011, from ACTAFFACT: http://www.ofai.at/~stefan.rank/StefanRank-AAAThesis.pdf

Rank, S., & Petta, P. (2007). From ActAffAct to BehBehBeh: Increasing Affective Detail in a Story-World. (C. M., & D. S., Eds.) *Virtual Storytelling: Fourth International Conference (ICVS 2007), LNCS 4871,* 206-209.

Rickett, J. (2009). *New Look for the Short Story.* (Guardian News and Media Limited) Retrieved September 21, 2011, from WritingNews.org: http://www.guardian.co.uk/books/2008/mar/24/news.uknews

Rohrer, J. (2010). *Sleep is Death Shannon Galvin's Contributions.* Retrieved September 21, 2011, from Sleep is Death: http://www.sleepisdeath.net/galvin.php

Rollings, A., & Adams, E. (2003). *On Game Design.* New Riders Games.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence A Modern Approach* (Second Edition ed.). Pearson Education.

Sgouros, N. M. (1999). Dynamic Generation, Management and Resolution of Interactive Plots. *Artificial Intelligence, 107*(1), 29-62.

Shen, Z., Miao, C., Tao, X., & Gay, R. (2005). *Goal-oriented Modelling for Intelligent Software Agents.* Singapore: Nanyang Technological University.

Slack, J. (2009, November 02). *Terrain Example*. Retrieved March 10, 2011, from Blog of Josh "Renanse" Slack : http://blog.renanse.com/2009/11/terrain-example.html

Spector, W. (2007). *Next-Gen Storytelling Part One: What Makes a Story?* Retrieved September 21, 2011, from escapistmagazine.com: http://www.escapistmagazine.com/news/view/70852-Next-Gen-Storytelling-Part-One-What-Makes-a-Story

Stout Games. (2011). *Dinner Date*. Retrieved April 25, 2011, from http://thestoutgames.com/:DinnerDate

Szilas, N. (2008). *Interactive Drama: The story is in your hands...* Retrieved January 15, 2011, from http://www.idtension.com/

Tanner, C. C., Migdal, C. J., & Jones, M. T. (1998). The Clipmap: A Virtual Mipmap. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, (pp. 151-158). New York.

Thue, D., Bulitko, V., Spetch, M., & Wasylishen, E. (2007). Interactive Storytelling: A Player Modelling Approach. *The Third Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE 07)*. Stanford, CA: Stanford University.

Todorov, T. (1969). *Grammaire du Decameron*. Paris.

Tomaszewski, Z., & Binsted, K. (2006). A Reconstructed Neo-Aristotelian Theory of Interactive Drama. *Workshop on Computational Aesthetics: Artificial Intelligence Approaches to Beauty and Happiness, National Conference on Artificial Intelligence (AAAI)*.

Tošić, S., Radovanović, M., & Ivanović, M. (2008). APP: Agent Planning Package. *Advances in Intelligent and Distributed Computing - Studies in Computational Intelligence, 78*, 217-226.

Tozour, P. (2008). *Fixing Pathfinding Once and For All*. Retrieved March 18, 2011, from Game/AI: http://www.ai-blog.net/archives/000152.html

Tychsen, A., Hitchens, M., Brolund, T., & Kavakli, M. (2005). The Game Master. Sydney, Australia : Creativity & Cognition Studios Press.

Wee, N., & Seifert, L. (2001). *Vladimir Propp's Theories*, 1.0. (Brown University) Retrieved September 21, 2011, from PROPPIAN FAIRY TALE GENERATOR: http://www.brown.edu/Courses/FR0133/Fairytale_Generator/propp.html

West, M. (2008, August 6). *Random Scattering Creating Realistic Landscapes.* Retrieved September 21, 2011, from Gamasutra: http://www.gamasutra.com/view/feature/1648/random_scattering_creating_.php

Wheeler, L. K. (2004). *Freytag's Pyramid.* Retrieved September 21, 2011, from http://web.cn.edu/kwheeler/freytag.html

Young, S. (2009, April 14). *Procedural city, Part 2: Building Textures.* Retrieved September 21, 2011, from Twenty Sided: http://www.shamusyoung.com/twentysidedtale/?p=2954

APPENDIX

APPENDIX 1 – Test 1 Console Output

This is the output from the DISE console when the Cloak of Darkness test one was executed (see section 6.1.2). The output shows the initial facts along with the character's decision making process composed of goals, actions and the execution times for each turn.

```
18-Jul-2011 15:36:09 ac.ljmu.dise.engine.DISEExampleBase init
INFO: Display Vendor: NVIDIA Corporation
18-Jul-2011 15:36:09 ac.ljmu.dise.engine.DISEExampleBase init
INFO: Display Renderer: GeForce 8500 GT/PCI/SSE2
18-Jul-2011 15:36:09 ac.ljmu.dise.engine.DISEExampleBase init
INFO: Display Version: 3.2.0
18-Jul-2011 15:36:09 ac.ljmu.dise.engine.DISEExampleBase init
INFO: Shading Language Version: 1.50 NVIDIA via Cg compiler
18-Jul-2011 15:36:10 ac.ljmu.dise.engine.BMFontLoader <init>
INFO: defaultFont = DejaVu Sans Condensed-20-bold-regular
fact 0: (accessible e0 e1)
fact 1: (accessible e1 e0)
fact 2: (accessible e1 e2)
fact 3: (accessible e2 e1)
fact 4: (accessible b0 b1)
fact 5: (accessible b0 b2)
fact 6: (accessible b0 b3)
fact 7: (accessible b0 b4)
fact 8: (accessible b0 b5)
fact 9: (accessible b0 b6)
fact 10: (accessible b1 b0)
fact 11: (accessible b2 b0)
fact 12: (accessible b3 b0)
fact 13: (accessible b4 b0)
fact 14: (accessible b5 b0)
fact 15: (accessible b6 b0)
fact 16: (accessible b1 b2)
fact 17: (accessible b2 b1)
fact 18: (accessible b3 b4)
fact 19: (accessible b4 b3)
fact 20: (accessible b4 b5)
fact 21: (accessible b5 b4)
fact 22: (accessible b5 b6)
fact 23: (accessible b6 b5)
fact 24: (accessible b4 b7)
fact 25: (accessible b7 b4)
fact 26: (accessible b3 b8)
fact 27: (accessible b8 b3)
fact 28: (accessible f0 f1)
fact 29: (accessible f1 f0)
fact 30: (accessible f1 f2)
fact 31: (accessible f2 f1)
fact 32: (accessible f2 f3)
fact 33: (accessible f3 f2)
fact 34: (accessible f2 f4)
fact 35: (accessible f4 f2)
fact 36: (accessible f2 f5)
fact 37: (accessible f5 f2)
fact 38: (accessible f3 f4)
fact 39: (accessible f4 f3)
```

```
     fact 40: (accessible f3 f6)
     fact 41: (accessible f6 f3)
     fact 42: (accessible f6 f7)
     fact 43: (accessible f7 f6)
     fact 44: (accessible c0 c1)
     fact 45: (accessible c1 c0)
     fact 46: (accessible c1 c2)
     fact 47: (accessible c2 c1)
     fact 48: (accessible f1 b0)
     fact 49: (accessible b0 f1)
     fact 50: (accessible e1 f1)
     fact 51: (accessible f1 e1)
     fact 52: (accessible c1 f0)
     fact 53: (accessible f0 c1)
     fact 54: (at npc1 e0)
     fact 55: (at npc2 c1)
     fact 56: (at npc3 e1)
     fact 57: (at hook c0)
     fact 58: (at message b6)
     fact 59: (wearing npc1 cloak)
     fact 60: (iswearable cloak)
     fact 61: (ishangable hook)
     fact 62: (isreadable message)
     fact 63: (islit f0)
     fact 64: (islit f1)
     fact 65: (islit f2)
     fact 66: (islit f3)
     fact 67: (islit f4)
     fact 68: (islit f5)
     fact 69: (islit f6)
     fact 70: (islit f7)
     fact 71: (islit c0)
     fact 72: (islit c1)
     fact 73: (islit c2)
     fact 74: (islit e0)
     fact 75: (islit e1)
     fact 76: (islit e2)
     fact 77: (islit b0)
     fact 78: (islit b1)
     fact 79: (islit b2)
     fact 80: (islit b3)
     fact 81: (islit b4)
     fact 82: (islit b5)
     fact 83: (islit b6)
     fact 84: (islit b7)
     fact 85: (islit b8)
     fact 86: (= (steps-used npc1) 0)
     fact 87: (= (steps-used npc2) 0)
     fact 88: (= (steps-used npc3) 0)
     fact 89: (= (AngerFear npc1) 0)
     fact 90: (= (AngerFear npc2) 0)
     fact 91: (= (AngerFear npc3) 0)
     object 91: e0 e1 e2 - location
     object 91: c0 c1 c2 - location
     object 91: b0 b1 b2 b3 b4 b5 b6 b7 b8 - location
     object 91: f0 f1 f2 f3 f4 f5 f6 f7 - location
     object 91: npc1 - character
     object 91: npc2 - character
     object 91: npc3 - character
     object 91: bob - character
     object 91: cloak - item
     object 91: hook - dynamic
     object 91: message - dynamic
     object 91: there-is-cloak - information
     18-Jul-2011 15:36:12
com.ardor3d.extension.model.collada.jdom.ColladaAnimUtils buildAnimations
     WARNING: No element-joint mapping found for element: <node id="IK_Chain01"
name="IK_Chain01"></node>

     Check if element is of type JOINT.
     18-Jul-2011 15:36:12
```

```
com.ardor3d.extension.model.collada.jdom.ColladaAnimUtils buildAnimations
     WARNING: No element-joint mapping found for element: <node id="IK_Chain02"
name="IK_Chain02"></node>

     Check if element is of type JOINT.
     18-Jul-2011 15:36:12
com.ardor3d.extension.model.collada.jdom.ColladaAnimUtils buildAnimations
     WARNING: No element-joint mapping found for element: <node id="IK_Chain04"
name="IK_Chain04"></node>

     Check if element is of type JOINT.
     18-Jul-2011 15:36:12
com.ardor3d.extension.model.collada.jdom.ColladaAnimUtils buildAnimations
     WARNING: No element-joint mapping found for element: <node id="IK_Chain03"
name="IK_Chain03"></node>

     Check if element is of type JOINT.
     Importing: URLResourceSource
[url=file:/C:/Workspaces/Homura21GamingWorkspace/CharacterEngineTest/bin/ac/ljmu/
dise/media/models/collada%2Fanimegirl%2Fanimegirl.dae, type=.dae]
     Took 1394 ms
     WARNING: Found unknown Windows version: Windows 7
     Attempting to use default windows plug-in.
     Loading: net.java.games.input.DirectAndRawInputEnvironmentPlugin

     Scene Trigger Test:

        testing precondition: (haveread npc1 message)
       ...not found - false

        testing precondition: (haveitem npc1 cloak)
       ...not found - false

     ["npc1" turn 0]
     find plan for npc1: goal = (haveread npc1 message)
     New 'problem.pddl' file Data Saved!
     C:\Users\cmpscoop\AppData\Local\Temp\problem1494506865186804637.pddl
     C:/Workspaces/Homura21GamingWorkspace/CharacterEngineTest/bin/ac/ljmu/dise/m
edia/planners/MetricFF/ff -o
C:/Workspaces/Homura21GamingWorkspace/CharacterEngineTest/bin/ac/ljmu/dise/media/
planners/MetricFF/domain.pddl -f
C:/Users/cmpscoop/AppData/Local/Temp/problem1494506865186804637.pddl
     Planner Output:
     0: GOTO NPC1 E0 E1
     1: GOTO NPC1 E1 F1
     2: GOTO NPC1 F1 B0
     3: TAKE-OFF NPC1 CLOAK
     4: GOTO NPC1 B0 F1
     5: GOTO NPC1 F1 F0
     6: GOTO NPC1 F0 C1
     7: GOTO NPC1 C1 C0
     8: HANG-UP NPC1 CLOAK HOOK C0
     9: GOTO NPC1 C0 C1
     10: GOTO NPC1 C1 F0
     11: GOTO NPC1 F0 F1
     12: GOTO NPC1 F1 B0
     13: GOTO NPC1 B0 B6
     14: READ NPC1 MESSAGE B6
      found plan for (haveread npc1 message): true
     Clear Action Lists!
       added action: GOTO NPC1 E0 E1 to plan list.
       added action: GOTO NPC1 E1 F1 to plan list.
       added action: GOTO NPC1 F1 B0 to plan list.
       added action: TAKE-OFF NPC1 CLOAK to plan list.
       added action: GOTO NPC1 B0 F1 to plan list.
       added action: GOTO NPC1 F1 F0 to plan list.
       added action: GOTO NPC1 F0 C1 to plan list.
       added action: GOTO NPC1 C1 C0 to plan list.
       added action: HANG-UP NPC1 CLOAK HOOK C0 to plan list.
       added action: GOTO NPC1 C0 C1 to plan list.
       added action: GOTO NPC1 C1 F0 to plan list.
```

```
   added action: GOTO NPC1 F0 F1 to plan list.
   added action: GOTO NPC1 F1 B0 to plan list.
   added action: GOTO NPC1 B0 B6 to plan list.
   added action: READ NPC1 MESSAGE B6 to plan list.
   moved action: GOTO NPC1 E0 E1 to exe list.
   current action: GOTO NPC1 E0 E1
 added fact (at npc1 e1)
 removed fact (at npc1 e0)
 updated fact (= (steps-used npc1) 1) old value=0 + increase=1
    WHEN 0:(wearing npc1 cloak)
   ...found - true
 THEN 0:(not (islit e1))
 removed fact (islit e1)
 THEN 1:(islit e0)
 added fact (islit e0)
    WHEN 0:(haveitem npc1 cloak)
   ...not found - false
    'when' is false, so not added fact/s:  (not (islit e1)) (islit e0)
 npc1 performs GOTO NPC1 E0 E1
   completed action: GOTO NPC1 E0 E1
   (so removed from exe list)
 End Turn!
  This turn took 47935731 nano seconds!

 Scene Trigger Test:
    testing precondition: (haveread npc1 message)
   ...not found - false

    testing precondition: (haveitem npc1 cloak)
   ...not found - false
 ["npc1" turn 1]
   current goal = top goal in list!
   action exe list IS empty!
   action plan list NOT empty!
    moved action: GOTO NPC1 E1 F1 to exe list.
    testing precondition: (currentcharacter npc1)
   ...found - true
    testing precondition: (at npc1 e1)
   ...found - true
    testing precondition: (accessible e1 f1)
   ...found - true
   [all action preconditions true!]
 added fact (at npc1 f1)
 removed fact (at npc1 e1)
 updated fact (= (steps-used npc1) 2) old value=1 + increase=1
    WHEN 0:(wearing npc1 cloak)
   ...found - true
 THEN 0:(not (islit f1))
 removed fact (islit f1)
 THEN 1:(islit e1)
 added fact (islit e1)
    WHEN 0:(haveitem npc1 cloak)
   ...not found - false
    'when' is false, so not added fact/s:  (not (islit f1)) (islit e1)
 npc1 performs GOTO NPC1 E1 F1
   current action: GOTO NPC1 E1 F1
   completed action: GOTO NPC1 E1 F1
   (so removed from exe list)
 End Turn!
  This turn took 1618741 nano seconds!

 Scene Trigger Test:

    testing precondition: (haveread npc1 message)
   ...not found - false

    testing precondition: (haveitem npc1 cloak)
   ...not found - false

 ["npc1" turn 2]
   current goal = top goal in list!
```

```
 action exe list IS empty!
 action plan list NOT empty!
  moved action: GOTO NPC1 F1 B0 to exe list.

  testing precondition: (currentcharacter npc1)
 ...found - true
  testing precondition: (at npc1 f1)
 ...found - true
  testing precondition: (accessible f1 b0)
 ...found - true
 [all action preconditions true!]

added fact (at npc1 b0)
removed fact (at npc1 f1)
updated fact (= (steps-used npc1) 3) old value=2 + increase=1
   WHEN 0:(wearing npc1 cloak)
  ...found - true
THEN 0:(not (islit b0))
removed fact (islit b0)
THEN 1:(islit f1)
added fact (islit f1)
   WHEN 0:(haveitem npc1 cloak)
  ...not found - false
   'when' is false, so not added fact/s:  (not (islit b0)) (islit f1)
npc1 performs GOTO NPC1 F1 B0
  current action: GOTO NPC1 F1 B0
  completed action: GOTO NPC1 F1 B0
  (so removed from exe list)
End Turn!
 This turn took 1424063 nano seconds!

Scene Trigger Test:

   testing precondition: (haveread npc1 message)
  ...not found - false

   testing precondition: (haveitem npc1 cloak)
  ...not found - false

["npc1" turn 3]
  current goal = top goal in list!
  action exe list IS empty!
  action plan list NOT empty!
   moved action: TAKE-OFF NPC1 CLOAK to exe list.

   testing precondition: (currentcharacter npc1)
  ...found - true
   testing precondition: (wearing npc1 cloak)
  ...found - true
  [all action preconditions true!]

added fact (haveitem npc1 cloak)
removed fact (wearing npc1 cloak)
npc1 performs TAKE-OFF NPC1 CLOAK
  current action: TAKE-OFF NPC1 CLOAK
  completed action: TAKE-OFF NPC1 CLOAK
  (so removed from exe list)
End Turn!
 This turn took 428377 nano seconds!

Scene Trigger Test:

   testing precondition: (haveread npc1 message)
  ...not found - false

   testing precondition: (haveitem npc1 cloak)
  ...found - true
TRIGGERED NEXT SCENE: act3
running scene story mods..
this scene has no story mods
```

```
["npc1" turn 4]
  current goal = top goal in list!
  action exe list IS empty!
  action plan list NOT empty!
   moved action: GOTO NPC1 B0 F1 to exe list.

   testing precondition: (currentcharacter npc1)
  ...found - true
   testing precondition: (at npc1 b0)
  ...found - true
   testing precondition: (accessible b0 f1)
  ...found - true
  [all action preconditions true!]

added fact (at npc1 f1)
removed fact (at npc1 b0)
updated fact (= (steps-used npc1) 4) old value=3 + increase=1
   WHEN 0:(wearing npc1 cloak)
  ...not found - false
   'when' is false, so not added fact/s:  (not (islit f1)) (islit b0)
   WHEN 0:(haveitem npc1 cloak)
  ...found - true
THEN 0:(not (islit f1))
removed fact (islit f1)
THEN 1:(islit b0)
added fact (islit b0)
npc1 performs GOTO NPC1 B0 F1
  current action: GOTO NPC1 B0 F1
  completed action: GOTO NPC1 B0 F1
  (so removed from exe list)
End Turn!
 This turn took 1169352 nano seconds!

Scene Trigger Test:
this scene has no story triggers

["npc1" turn 5]
  current goal = top goal in list!
  action exe list IS empty!
  action plan list NOT empty!
   moved action: GOTO NPC1 F1 F0 to exe list.

   testing precondition: (currentcharacter npc1)
  ...found - true
   testing precondition: (at npc1 f1)
  ...found - true
   testing precondition: (accessible f1 f0)
  ...found - true
  [all action preconditions true!]

added fact (at npc1 f0)
removed fact (at npc1 f1)
updated fact (= (steps-used npc1) 5) old value=4 + increase=1
   WHEN 0:(wearing npc1 cloak)
  ...not found - false
   'when' is false, so not added fact/s:  (not (islit f0)) (islit f1)
   WHEN 0:(haveitem npc1 cloak)
  ...found - true
THEN 0:(not (islit f0))
removed fact (islit f0)
THEN 1:(islit f1)
added fact (islit f1)
npc1 performs GOTO NPC1 F1 F0
  current action: GOTO NPC1 F1 F0
  completed action: GOTO NPC1 F1 F0
  (so removed from exe list)
End Turn!
 This turn took 1220381 nano seconds!

Scene Trigger Test:
this scene has no story triggers
```

```
["npc1" turn 6]
  current goal = top goal in list!
  action exe list IS empty!
  action plan list NOT empty!
   moved action: GOTO NPC1 F0 C1 to exe list.

   testing precondition: (currentcharacter npc1)
  ...found - true
   testing precondition: (at npc1 f0)
  ...found - true
   testing precondition: (accessible f0 c1)
  ...found - true
  [all action preconditions true!]

added fact (at npc1 c1)
removed fact (at npc1 f0)
updated fact (= (steps-used npc1) 6) old value=5 + increase=1
   WHEN 0:(wearing npc1 cloak)
  ...not found - false
   'when' is false, so not added fact/s:  (not (islit c1)) (islit f0)
   WHEN 0:(haveitem npc1 cloak)
  ...found - true
THEN 0:(not (islit c1))
removed fact (islit c1)
THEN 1:(islit f0)
added fact (islit f0)
npc1 performs GOTO NPC1 F0 C1
  current action: GOTO NPC1 F0 C1
  completed action: GOTO NPC1 F0 C1
  (so removed from exe list)
End Turn!
 This turn took 1069870 nano seconds!

Scene Trigger Test:
this scene has no story triggers

["npc1" turn 7]
  current goal = top goal in list!
  action exe list IS empty!
  action plan list NOT empty!
   moved action: GOTO NPC1 C1 C0 to exe list.

   testing precondition: (currentcharacter npc1)
  ...found - true
   testing precondition: (at npc1 c1)
  ...found - true
   testing precondition: (accessible c1 c0)
  ...found - true
  [all action preconditions true!]

added fact (at npc1 c0)
removed fact (at npc1 c1)
updated fact (= (steps-used npc1) 7) old value=6 + increase=1
   WHEN 0:(wearing npc1 cloak)
  ...not found - false
   'when' is false, so not added fact/s:  (not (islit c0)) (islit c1)
   WHEN 0:(haveitem npc1 cloak)
  ...found - true
THEN 0:(not (islit c0))
removed fact (islit c0)
THEN 1:(islit c1)
added fact (islit c1)
npc1 performs GOTO NPC1 C1 C0
  current action: GOTO NPC1 C1 C0
  completed action: GOTO NPC1 C1 C0
  (so removed from exe list)
End Turn!
 This turn took 1240963 nano seconds!

Scene Trigger Test:
```

```
this scene has no story triggers

["npc1" turn 8]
  current goal = top goal in list!
  action exe list IS empty!
  action plan list NOT empty!
   moved action: HANG-UP NPC1 CLOAK HOOK C0 to exe list.

   testing precondition: (currentcharacter npc1)
  ...found - true
   testing precondition: (haveitem npc1 cloak)
  ...found - true
   testing precondition: (at npc1 c0)
  ...found - true
   testing precondition: (at hook c0)
  ...found - true
   testing precondition: (ishangable hook)
  ...found - true
  [all action preconditions true!]

added fact (on hook cloak)
removed fact (haveitem npc1 cloak)
npc1 performs HANG-UP NPC1 CLOAK HOOK C0
  current action: HANG-UP NPC1 CLOAK HOOK C0
  completed action: HANG-UP NPC1 CLOAK HOOK C0
  (so removed from exe list)
End Turn!
 This turn took 636348 nano seconds!

Scene Trigger Test:
this scene has no story triggers

["npc1" turn 9]
  current goal = top goal in list!
  action exe list IS empty!
  action plan list NOT empty!
   moved action: GOTO NPC1 C0 C1 to exe list.

   testing precondition: (currentcharacter npc1)
  ...found - true
   testing precondition: (at npc1 c0)
  ...found - true
   testing precondition: (accessible c0 c1)
  ...found - true
  [all action preconditions true!]

added fact (at npc1 c1)
removed fact (at npc1 c0)
updated fact (= (steps-used npc1) 8) old value=7 + increase=1
   WHEN 0:(wearing npc1 cloak)
  ...not found - false
   'when' is false, so not added fact/s:  (not (islit c1)) (islit c0)
   WHEN 0:(haveitem npc1 cloak)
  ...not found - false
   'when' is false, so not added fact/s:  (not (islit c1)) (islit c0)
npc1 performs GOTO NPC1 C0 C1
  current action: GOTO NPC1 C0 C1
  completed action: GOTO NPC1 C0 C1
  (so removed from exe list)
End Turn!
 This turn took 1093882 nano seconds!

Scene Trigger Test:
this scene has no story triggers

["npc1" turn 10]
  current goal = top goal in list!
  action exe list IS empty!
  action plan list NOT empty!
   moved action: GOTO NPC1 C1 F0 to exe list.
```

```
    testing precondition: (currentcharacter npc1)
   ...found - true
    testing precondition: (at npc1 c1)
   ...found - true
    testing precondition: (accessible c1 f0)
   ...found - true
   [all action preconditions true!]

 added fact (at npc1 f0)
 removed fact (at npc1 c1)
 updated fact (= (steps-used npc1) 9) old value=8 + increase=1
    WHEN 0:(wearing npc1 cloak)
   ...not found - false
    'when' is false, so not added fact/s:  (not (islit f0)) (islit c1)
    WHEN 0:(haveitem npc1 cloak)
   ...not found - false
    'when' is false, so not added fact/s:  (not (islit f0)) (islit c1)
 npc1 performs GOTO NPC1 C1 F0
   current action: GOTO NPC1 C1 F0
   completed action: GOTO NPC1 C1 F0
   (so removed from exe list)
 End Turn!
  This turn took 1067725 nano seconds!

 Scene Trigger Test:
 this scene has no story triggers

 ["npc1" turn 11]
   current goal = top goal in list!
   action exe list IS empty!
   action plan list NOT empty!
    moved action: GOTO NPC1 F0 F1 to exe list.

    testing precondition: (currentcharacter npc1)
   ...found - true
    testing precondition: (at npc1 f0)
   ...found - true
    testing precondition: (accessible f0 f1)
   ...found - true
   [all action preconditions true!]

 added fact (at npc1 f1)
 removed fact (at npc1 f0)
 updated fact (= (steps-used npc1) 10) old value=9 + increase=1
    WHEN 0:(wearing npc1 cloak)
   ...not found - false
    'when' is false, so not added fact/s:  (not (islit f1)) (islit f0)
    WHEN 0:(haveitem npc1 cloak)
   ...not found - false
    'when' is false, so not added fact/s:  (not (islit f1)) (islit f0)
 npc1 performs GOTO NPC1 F0 F1
   current action: GOTO NPC1 F0 F1
   completed action: GOTO NPC1 F0 F1
   (so removed from exe list)
 End Turn!
  This turn took 1045857 nano seconds!

 Scene Trigger Test:
 this scene has no story triggers

 ["npc1" turn 12]
   current goal = top goal in list!
   action exe list IS empty!
   action plan list NOT empty!
    moved action: GOTO NPC1 F1 B0 to exe list.

    testing precondition: (currentcharacter npc1)
   ...found - true
    testing precondition: (at npc1 f1)
   ...found - true
    testing precondition: (accessible f1 b0)
```

```
   ...found - true
   [all action preconditions true!]

 added fact (at npc1 b0)
 removed fact (at npc1 f1)
 updated fact (= (steps-used npc1) 11) old value=10 + increase=1
    WHEN 0:(wearing npc1 cloak)
   ...not found - false
    'when' is false, so not added fact/s:  (not (islit b0)) (islit f1)
    WHEN 0:(haveitem npc1 cloak)
   ...not found - false
    'when' is false, so not added fact/s:  (not (islit b0)) (islit f1)
 npc1 performs GOTO NPC1 F1 B0
   current action: GOTO NPC1 F1 B0
   completed action: GOTO NPC1 F1 B0
   (so removed from exe list)
 End Turn!
  This turn took 1114465 nano seconds!

 Scene Trigger Test:
 this scene has no story triggers

 ["npc1" turn 13]
   current goal = top goal in list!
   action exe list IS empty!
   action plan list NOT empty!
    moved action: GOTO NPC1 B0 B6 to exe list.

    testing precondition: (currentcharacter npc1)
   ...found - true
    testing precondition: (at npc1 b0)
   ...found - true
    testing precondition: (accessible b0 b6)
   ...found - true
   [all action preconditions true!]

 added fact (at npc1 b6)
 removed fact (at npc1 b0)
 updated fact (= (steps-used npc1) 12) old value=11 + increase=1
    WHEN 0:(wearing npc1 cloak)
   ...not found - false
    'when' is false, so not added fact/s:  (not (islit b6)) (islit b0)
    WHEN 0:(haveitem npc1 cloak)
   ...not found - false
    'when' is false, so not added fact/s:  (not (islit b6)) (islit b0)
 npc1 performs GOTO NPC1 B0 B6
   current action: GOTO NPC1 B0 B6
   completed action: GOTO NPC1 B0 B6
   (so removed from exe list)
 End Turn!
  This turn took 1067297 nano seconds!

 Scene Trigger Test:
 this scene has no story triggers

 ["npc1" turn 14]
   current goal = top goal in list!
   action exe list IS empty!
   action plan list NOT empty!
    moved action: READ NPC1 MESSAGE B6 to exe list.

    testing precondition: (currentcharacter npc1)
   ...found - true
    testing precondition: (at npc1 b6)
   ...found - true
    testing precondition: (at message b6)
   ...found - true
    testing precondition: (isreadable message)
   ...found - true
    testing precondition: (islit b6)
   ...found - true
```

```
   [all action preconditions true!]

 added fact (haveread npc1 message)
 npc1 performs READ NPC1 MESSAGE B6
   current action: READ NPC1 MESSAGE B6
   completed action: READ NPC1 MESSAGE B6
   (so removed from exe list)
 End Turn!
  This turn took 498701 nano seconds!

 Scene Trigger Test:
 this scene has no story triggers

 ["npc1" turn 15]
   current goal = top goal in list!
   action exe list IS empty!
   action plan list IS empty!
 GOAL have read message COMPLETE!
 End Turn!
  This turn took 154370 nano seconds!

 Scene Trigger Test:
 this scene has no story triggers

 ["npc1" turn 16]
 no more goals to complete - set wandering
 npc1 is WANDERING!
 End Turn!
  This turn took 92193 nano seconds!

 Scene Trigger Test:
 this scene has no story triggers

 ["npc1" turn 17]
 no more goals to complete - set wandering
 npc1 is WANDERING!
 End Turn!
  This turn took 91765 nano seconds!

 Scene Trigger Test:
 this scene has no story triggers

 ["npc1" turn 18]
 no more goals to complete - set wandering
 npc1 is WANDERING!
 End Turn!
  This turn took 80615 nano seconds!

 Scene Trigger Test:
 this scene has no story triggers

 ["npc1" turn 19]
 no more goals to complete - set wandering
 npc1 is WANDERING!
 End Turn!
  This turn took 75470 nano seconds!
 Save results
```

APPENDIX 2 – DISE Editor Tutorial

This section contains a tutorial to guide users through the creation process for new stories. A DISE story can be created using the following steps:

1. Adding Predicate Definitions
2. Adding Action Definitions
3. Adding Objects
4. Adding Personality Variables
5. Defining Characters
6. Creating the World
7. Defining the Initial World State
8. Creating the Scene Structure

The first five of these can be used to create re-useable assets, such as predicate definitions, actions, props, characters and locations. The initial state and scene structure are used to choose which of these assets from the library are used, which state they start in and how they change over time or reactively to the player.

1. Adding Predicate Definitions

Predicates are used to represent the state of all objects (nouns) in the story domain and are the main components in the Fact Database along with the list of objects in the story (which are used as the subject parameters for the predicates).

Example of the code produced for the 'at' and 'have item' predicates are given below:

```
 (at ?who/what - (either character item dynamic static)
?where - location)
 (haveitem ?who - character ?what - item)
```

New predicates like these can be created without writing code using the 'predicate editor' window by typing a unique name into the large box and clicking the add button. They can also be removed using their delete (X) button. Once created, they need to have their parameters and parameter types set. This is achieved by clicking the corresponding edit button and adding a row for each parameter. Parameters are automatically named x, y, z, w, etc. to make things easier. The Author then just needs to put a tick against each type that the parameter shares. Figure 1 below shows how the 'at' predicate in the previous section was created. The first predicate (who/what) has types (either character item dynamic static) and the second (where) can only be a location. When finished the save button can be clicked to save and close the popup. Defining types limits what objects can be set as the subject parameter of the predicates and also helps when defining actions, goals and initial world state in the other editor panels by narrowing down the available choices in the drop down menus. The DISE editor contains some example predicate

definitions, but it is up to the Story Author to create any extra definitions that they require for any new states. In this example for the 'Cloak of Darkness' story we created some extra predicates, such as: 'wearing' and 'have read'. Wearing is the state of a character and the wearable item which they are wearing. 'Have read' is the state of a character relating to readable interactive object (dynamic environment or item).

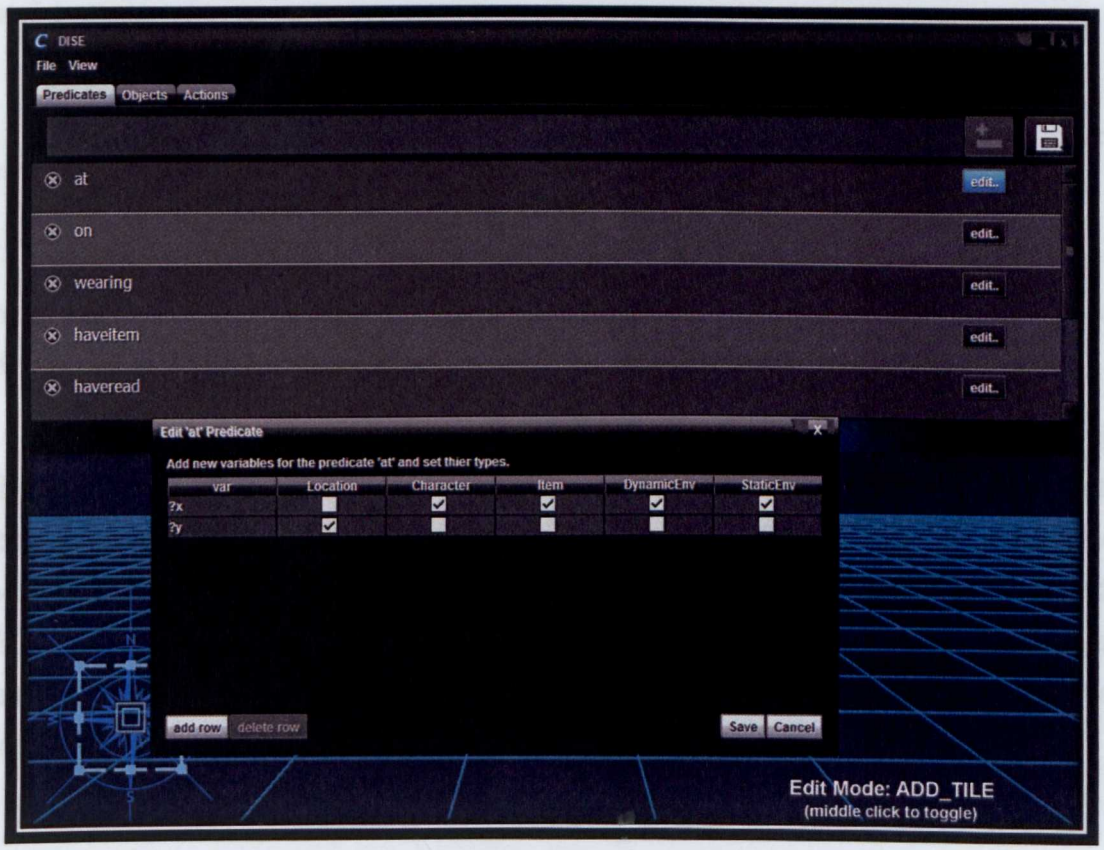


Figure 1: Predicate Editor Panel

2. Adding Action Definitions

To create the full description of an action when playing a story the user needs to form a sentence using a set grammar that humans can understand and that the computer can interpret. To achieve this each action needs a verb to describe the action, immediately followed by an agent to carry out the action. The next part of the sentence can vary depending on the subject of the action and the objects needed to carry it out. This third word will be a noun of type character, location, prop, or dynamic object. The next word is an optional noun, only used by some actions that need to combine more objects that interact with each other. Any other nouns can also be added after this, stretching the length of the action sentence to a number of words. The last word is always the location where the character needs to be to perform this action, which allows the characters to plan their movement and path-find to the correct area.

The final action sentence format is then:

VERB, CHARACTER, NOUN, (OPTIONAL) NOUN, [...], LOCATION

A basic action could consist of only three words:

```
take-off (?who - character ?what - item)
take-off (npc1, cloak)
```

Where npc1 is a computer controlled character and cloak is an item that is currently worn by npc1.

An example of a more complex action sentence that uses five words would be:

```
hang-up (?who - character ?what - item ?hangOnWhat - dynamic
?where - location)
hang-up(npc1, cloak, hook, c1)
```

Where npc1 is a computer controlled character, cloak is an item that can be hung up, hook is an interactive (dynamic) prop which can have an item hanging on it and c1 is sub-node 1 of the cloakroom's navigation mesh.

Using the actions panel in the editor new verbs can be created fairly easily and without knowing the exact PDDL code syntax. Figure 2 below shows the prototype action editor and the popup windows used to create and edit actions with their parameters, preconditions and effects. Similar to predicates there are some actions already listed in DISE, but more can be added by the Story Author to meet the requirements of their story's scenario. At this stage we are just defining the action verbs and if/when they are relevant, along with their effects.

The actions are usually made up of three parts:

- **Parameters** – Usually the 'Name' and 'Type' of an object, character, or location that are used by the action.
- **Preconditions** – Requirements that need to be met (i.e. states of the world that must equal true) to perform the action.
- **Effects** – Post conditions and modifications to the world's states after the action is carried out.

The following code is produced from the information input in the diagram (Figure 2) below:

```
(:action scare
 :parameters (?x-character ?y-character ?z-location)
 :precondition (and (at ?x ?z)(at ?y ?z)(not(= ?x ?y)))
 :effect (and (decrease (AngerFear ?y) 1))
)
```

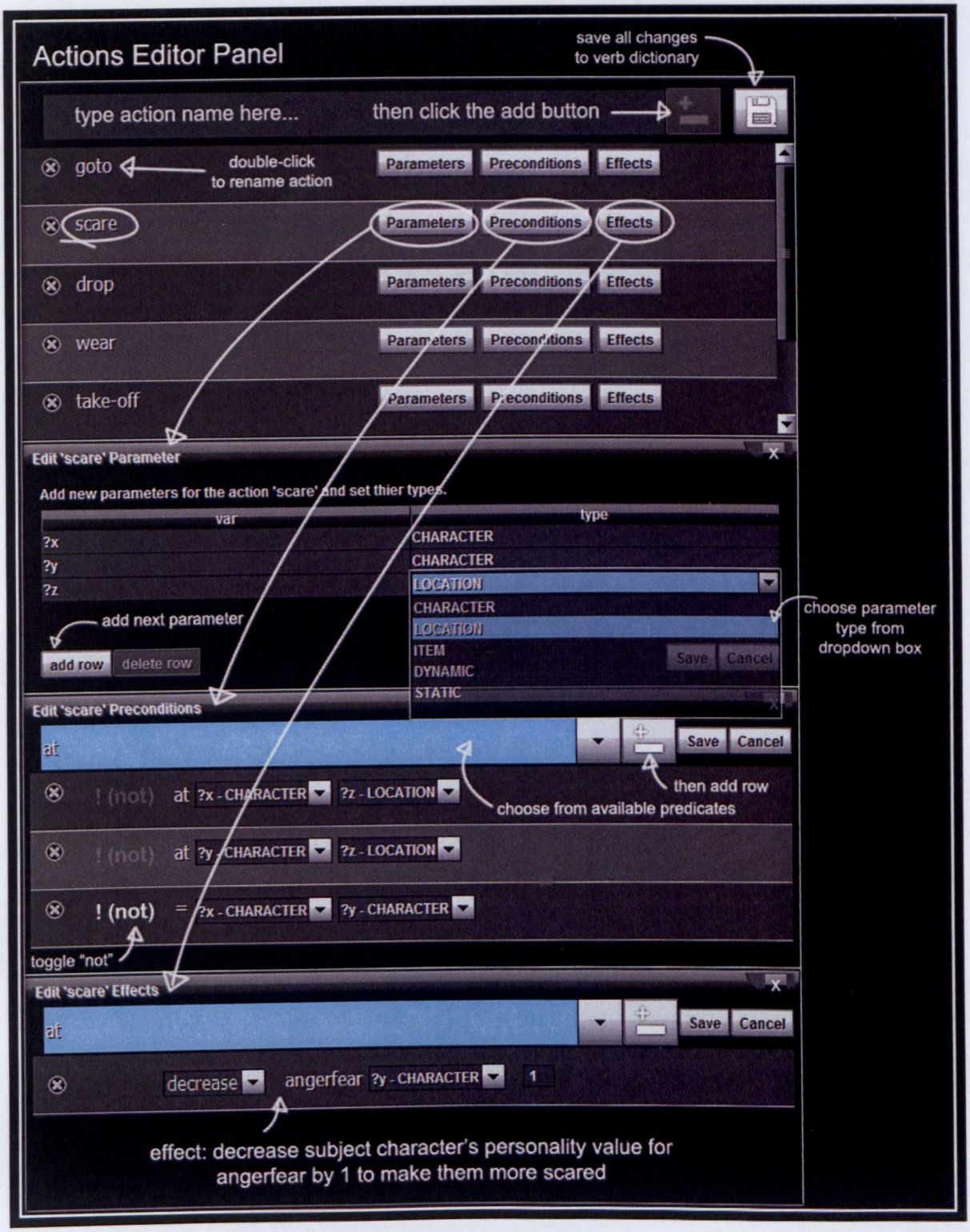


Figure 2: Action Editor Panel

To create this code first the action name is typed in the top text box and is then added using the 'add action' button. It will be appended to the scrolling list of all actions below and can be re-named by double-clicking its text label and changing the text. At first the parameters, preconditions and effects will be empty so each one needs to be filled in with the appropriate data by clicking the corresponding button to open a new window. The parameters section needs a list of the variables concerned with the action and their types. The 'add row' button adds a new variable which is automatically given a name (x, y, z, w,

etc.). The types can be selected using the dropdown box in the second column and selected parameters can be removed completely with the delete row button. To edit the preconditions the main dropdown is used to choose an available predicate for the particular precondition. Then pressing the add button will add the predicate to the list. The editor will automatically filter and list any parameters that are useable in the new predicate row. These can be changed to any parameter/type available to make the correct precondition test. Clicking save will save and exit the precondition popup window. The final effects can be added with the effects button, which opens up a similar window to the preconditions. Here effect predicates and their parameters/types can be defined and saved (Figure 2).

To create the actions the Story Author needs to think logically about the verbs meaning and what things are involved with the action. Every object used in the action needs to be listed as a parameter.

For every action that is carried out we want a set of reactive goals to be available along with the PDDL effects state change rules, if deemed necessary for that action. To provide this the effect section will also contain additional rules that describe how non-player characters (NPC) should react to a player's actions and determine which goals they will choose. The NPCs will be sequenced using the concept of roles. For each different role there are four main elements to consider. A role can be anything that a story builder defines in a verb/action and more roles and reactions could be added later if they are needed. Some actions have a direct subject to them that can be easily defined; other roles could be characters that passively observed the action and want to intervene.

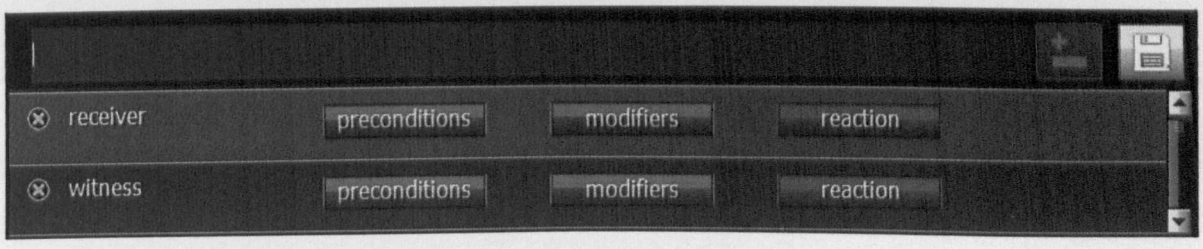


Figure 3: Action Roles Edit Panel

Click the 'add role' button for the desired action. This role is designed to specify a particular character or group of characters, change their personality model in reaction to the event and then make them choose a response goal based on their personality. For each role a name for that role is entered in a similar way to creating a new action e.g. 'Role – receiver' (Figure 3 above). Next the role needs preconditions, entered in exactly the same way as the action preconditions mentioned above. The next panel is to edit the personality modifiers. These can be a simple change in a value or an if-then statement to test a value then apple a modifier. The final part is another if-then statement similar to the preconditions panel, but with the outcome adding a new goal to the character in the

current role. For an example of the code generated by the input panels consider the table below:

Effect:

NPC Roles:

5. **Rules to describe which character should react to the event and which role they fit into.**
6. **The Modifiers that change the personality variables of the character in the specific role, which reflects how they feel about the current action and how it physically affected them.**
7. **A list of Reaction Goals to choose from and act out.**
8. **A Choice Formula that allows the character to choose the most suitable verb from the aforementioned list according to their updated personality model.**

Example:

```
Role:
• Receiver - Character
Modify:
• if Likes(Receiver, Item) then Likes(Receiver, Giver) + 0.1f
  * Amount_Receiver_Likes_Item
React:
• if Likes(Receiver, Item) then Thank(Receiver, Giver)
• if Likes(Receiver, Item) And Likes(Receiver, Giver) > 0.8f
  And In_Relationship(Receiver, Giver) then Kiss(Receiver,
  Giver)
• if !Likes(Receiver, Item) And !Likes(Receiver, Giver) then
  Give(Receiver, Giver, Item)
```

```
Role:
• Witness - Character
• Witness != Giver And Witness != Receiver And Witness See Give
Modify:
• Perceived_Generosity(Witness, Giver) + 0.1f * Item_Value
• If Likes(Witness, Item) And Jealous_Person(Witness) then
  Jealousy(Witness, Receiver) + 0.1f * Likes(Witness, Item)
• If Likes(Witness , Giver) And Jealous_Person(Witness) then
  Jealousy(Witness, Receiver) + 0.1f * Likes(Witness, Giver)
React:
• If Jealousy(Witness, Receiver)>0.9f then
  Snide_Comment(Witness, Receiver)
```

3. Adding Objects

The object PDDL definition, physics type and model file can all be easily input using this panel in the Story Editor. A new object can be created by typing its name into the top box and hitting the add button. This will create a new row in the objects list, which can be deleted or edited further. There are two drop down boxes allowing the story author to choose the type of the object and the physics type and a file browser allows the easy location and importing of Collada model files. Items can also be given an inventory block size and shape by highlighting grid squares if this is required in the story. These objects can then be positioned using the keyboard and mouse in the prop free drop editor to give them a position location and their exact vector data.

Figure 4: Object Edit Panel

4. Adding Personality Variables

In DISE the personality model only limits story authors to the five categories: First-order, Accordance, Second-order, Third-order and Mood variables. Within these fixed categories any new variables can be added to the models master list by a story author using the editor, as long as they have a unique identifier name. For example if the story was in the detective genre, the author may want to add a variable for 'detection' in the characters' first-order variables list. If an NPC has a high detection value they could be programmed in the current scene to inform the player that they have noticed a clue to

progress the story.

The default variables are split into their five types as follows:

- First-order variables

```
Honest
Virtuous
Powerful
Intelligent
Attractive
```

- Accordance variables

```
AccordHonesty
AccordVirtue
AccordPower
AccordIntelligence
AccordAttractive
```

- Second-order variables

```
PerHonest
PerVirtue
PerPowerful
PerIntelligent
PerAttractive
```

- Third-order variables

```
PerPerHonest
PerPerVirtue
PerPerPowerful
PerPerIntelligent
PerPerAttractive
```

- Moods (bi-polar +/-)

```
Passion/Disgust
Joy/Sadness
Anger/Fear
```

To add a new variable just scroll to the relevant category, name the new variable with a unique name and click the 'add new' button. All the values will be locked between zero and one, except for moods, which are between minus one and one. To remove an unwanted variable just click on the 'x' delete button next to it.

5. Defining characters

The character editor enables the easy creation of new characters. It has data entry user interfaces for:

- The character's initial personalities/moods to be set using individual sliders for each value.
- The model skin to represent the character to be loaded using the file browser.
- The character's initial position/location (similar to the Object Editor Panel).

- The initial state, such as items to be carried by each character, using a panel similar to the Init Editor Panel.
- The character's inventory contents and layout.
- The initial goals to be set for each character, using a panel similar to the Init Editor Panel.

6. Creating the World

The world editor allows the story authors to build a 3d environment or stage/set for their story to take place on. When a location is defined in the object editor panel clicking on the load model button will allow one of the world editors to run. Depending on the type environment required there are specific types of editor modes the Story Author can choose:

- **Block Mode** (Figure 6) – this allows the user to build up their world from textured blocks by left clicking the face of an existing block to create a new cloned block in the 90 degree compass direction (N, E, S, W) of the selected face. Using the right mouse button the block directly under the cursor can be removed.
- **Vertex Mode** (Figure 6) – this is selected by clicking the centre mouse button and allows the shape of the selected box face to be altered by moving single or multiple vertices using the **transform face tool** (Figure 5: Right), found on the HUD compass. Left clicking will move vertex points towards the camera and right clicking will move vertices away from the camera. A corner point will move only the selected corner vertex; the edge mid-pints markers will move both connecting corner vertices and the centre square will move all four face vertices. The blocks vertices are saved by storing the value of each point, along with its offset value (Figure 5: Left).
- In both block mode and vertex mode the texture coordinate values are also altered to tile the texture to accommodate changes in shape and size.

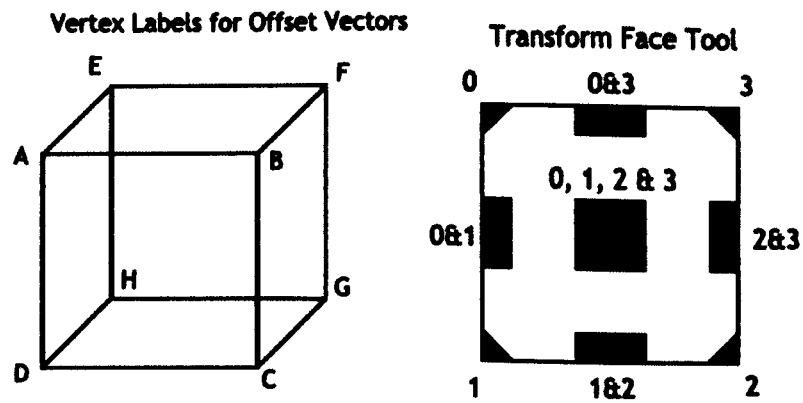


Figure 5: The Vertex Labels for Each Box and Transform Face Tool UI with Vertex Numbers

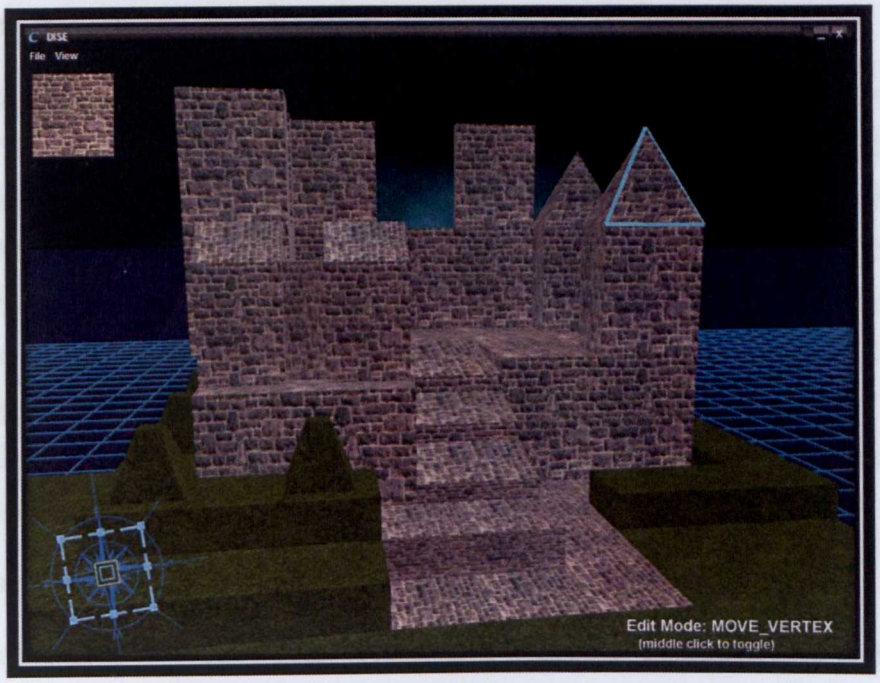


Figure 6: DISE World Editor Vertex Mode

- **Room Mode** (Figure 7) – this editor allows the user to draw squares on the floor plan grid by clicking and dragging with the left mouse button. The editor will then automatically calculate the surrounding walls for the room. The left mouse button adds rooms, or if clicked in an already existing room will extend that room; whilst the right mouse button will subtract the rectangular selection shape from any rooms on the grid. Pressing the space bar allows door frames/archways to be placed on a wall tile with an empty square either side, to create access points and join adjacent rooms and corridors.

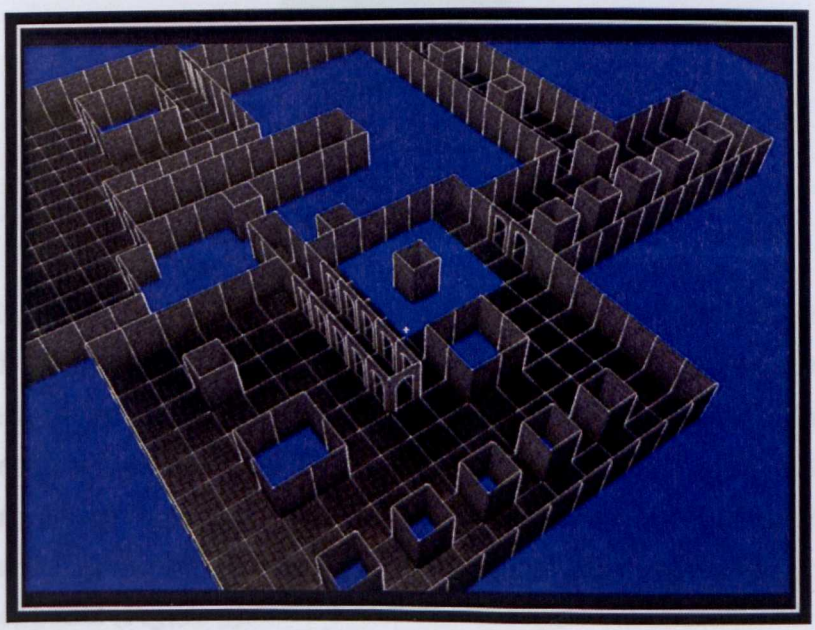


Figure 7: DISE World Editor Room Mode

7. Defining the Initial World State

The init editor panel displays a list of the initial state of the story world using predicates. The predicates can be chosen from the ones created in the predicate editor panel mentioned above. These can be deleted using their corresponding (x) buttons and added by choosing a predicate from the dropdown menu and clicking 'add'. The new predicate will then appear in the list and objects can be selected from the dropdown menus for each parameter. The predicates subject parameters will only list objects of the matching types, so no impossible facts can be added. For example 'have item' would only have two dropdown menus (one for each of its two parameters), containing only characters and items respectively.

8. Creating the Scene Structure

The Scene Editor is used to add a structure to the story using scenes linked together by StoryTriggers. The scene itself contains data to change the current state of the story world called a StoryMod.

To add a scene click the '+ new scene' button and move the new scene to the desired location (Figure 8). Double-click the scene to edit its name. This also brings up more options to edit the scene's contents or to add a new trigger.

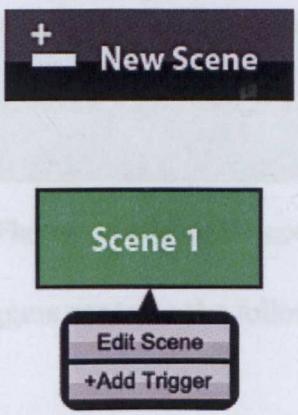


Figure 8: Creating a New Scene

If we add two more scenes and move them into position then select the first one again, the add trigger button can be selected. Next the Story Author needs to click on another scene to attach to the first one using the trigger connector. This will create a flexible link between them with an alarm clock icon in the middle. The clock will have a red 'X' over it at first to show that it is empty and can also be given a name in the same way as the scene earlier. Also if we double click the clock another window will open. This is the trigger manager panel (Figure 9).

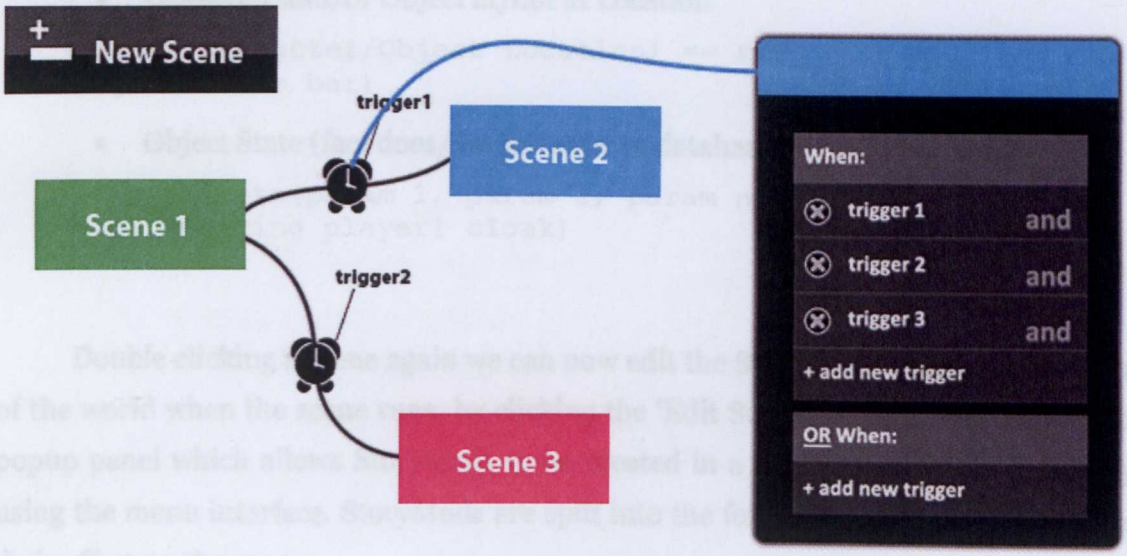


Figure 9: Story Editor and Scene Trigger Manager

Using the 'add new trigger panel' under the when heading multiple triggers can be chained together. The 'X' icon next to a trigger allows it to be removed when clicked. Using the 'Or When' heading allows multiple alternative triggers to be defined. These individual triggers can also be named and edited. These triggers can be set using the dropdown box and text field input methods similar to the preconditions in the action editor panel but relating to specific instances of an object (Figure 10).

Figure 10: Adding Triggers

The current list of story triggers contains the following types, but can also be expanded if necessary:

- Time (exactly t, after t, before t)

```
If time == t
If time > t
If time < t
```

- Personality

```
If (Attribute Character) == value
If (Attribute Character) < value
If (Attribute Character) > value
e.g. (=(AngerFear bob)0)
```

- Event

```
If (Verb Character Noun…) == (v, c, n…)
e.g. (scare player1 bob)
```

- Character and/or Object at/not at Location

```
If (at Character/Object Location) == t/f
e.g. (at bob bar)
```

- Object State (fact does/doesn't exist in database)

```
If predicate(param 1, param 2, param n) == t/f
e.g. (wearing player1 cloak)
```

Double clicking a scene again we can now edit the StoryMods which change the state of the world when the scene runs, by clicking the 'Edit Scene' button. This shows another popup panel which allows StoryMods to be created in a similar way to the StoryTriggers using the menu interface. StoryMods are split into the following categories, depending on their effect on the story:

- **Advance the story**
  - Progress time forwards, e.g. `(increase (time-days) 1).`
  - Display Message Text t, e.g. `"you wake up a day later and find out that your magic cloak has gone".`
  - Add/remove props, e.g. `remove: cloak.`
  - Add/remove facts, e.g. `add: (islit b0).`
  - Add/remove characters, e.g. `remove: npc1.`
  - Add/remove character goals e.g. `remove: (have-read npc1 message).`
  - Change character personality values, e.g. `(decrease (AngerFear npc1) 1).`
  - Display a Charisma scene animation. E.g. animated character dialogue explaining why the quest is important.
- **Create conflict**
  - Add/remove props, e.g. add things that hinder the player or remove helpful items that need to be reclaimed.
  - Change characters personality & goals to create conflict (see Develop a character below).
- **Introduce a character**
  - Character enter/leave story.
- **Develop a character**
  - Change personality variables, e.g. `(decrease (AngerFear ?npc1) 1)` or `(perAttractive ?npc1 - character ?npc2 - character).`
  - Give character a new goal, e.g. `(have-read npc1 message).`

- Display a Charisma scene animation.

- **Create suspense**
  - Add/remove facts, e.g. add `(islit b0)` would turn on the light in bar area zero.
  - Add/remove props.
  - Display Message Text t.
- **Giving out information**
  - Display Message Text t.
  - Display a Charisma scene animation.
- **Creating an atmosphere**
  - Add/remove facts, e.g. add `(islit b0)` would turn on the light in bar area zero.
  - Add/remove props.
  - Display Message Text t.
- **Develop the story's theme**
  - See advance the story above.

This base list allows some freedom to create a variety of stories for example: The long lost heir to the throne enters the story and has the goal to reclaim the kingdom. While the current evil ruler gains the new goal to kill the heir and has an increased value for anger.

The last part of the scene's data is the verbs list. This is a list of every verb defined in the action editor panel with a checkbox next to it. To allow an action to be used in the current scene mark it with a tick. There is also a select/deselect all button. Each new scene starts with the checked actions from the previous scene and can be modified from there to add and remove possible actions. By restricting certain actions the Story Author can shape the direction of the current scenes narrative.

Although a story requires a large amount of data for both logic and displaying graphics, the editors make it easier for someone who can't code or use advanced 3D Modelling packages to create something interesting and fun to play.