EXAMINING SUPPORT OF NARRATIVE SCRIPTING FOR SERIOUS GAMES

by

Marty Kauhanen

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

MASTER OF COMPUTER SCIENCE

School of Computer Science

at

CARLETON UNIVERSITY

Ottawa, Ontario
May, 2009

Your file Votre référence
ISBN: 978-0-494-52048-2
Our file Notre référence
ISBN: 978-0-494-52048-2

Table of Contents

List of Tables

List of Figures

Abstract

In this thesis, we explore the support required to build interactive narratives for serious games. Serious games typically are built to support learning, training or persuasion and have different requirements than other types of games, including shorter durations of game-play and much smaller development budgets. In addition, there is a need to support non-programmers in scripting serious games. We believe authoring tools can provide this support. We examine the literature and techniques for interactive narrative scripting and propose the basis of a pattern language for support. We employ techniques from human-computer interaction (HCI), borrowing from the approach used in Heuristic Evaluation in the application of the Cognitive Dimensions of Notations, to conduct a study to examine a successful commercial game development tool that supports interactive narrative components. We employ other techniques from HCI to conduct studies at a professional game development company. We use the principles of Contextual Design when observing and analyzing the work of the key contributors to the games. We perform both a Cognitive Walkthrough and usability study of the tools they use to integrate narrative content into their serious games. And finally, we present a model of support for scripting interactive narratives for serious games to guide further research.

Acknowledgements

First and foremost, I express my thanks to Dr. Robert Biddle for his continued direction, guidance and patience throughout this endeavor. I value his experience and insights, both in formal and informal settings, and acknowledge his dedication to his students, including yours truly.

I wish to express my appreciation to the management and employees at a particular Ottawa-area company for the opportunity to interact and observe professional game developers at work. Their patience and friendliness were much appreciated. Most of all, I admire their efforts towards opening up serious game development to a wider audience.

I thank the HOT Lab students for their camaraderie, with special thanks given to Minh Tran, who accompanied me on many site visits, and to Chris Eaket for his advice and creativity.

To my close friends: I appreciate your encouragement and understanding. In particular, I thank Joe and Jen Tereschuk for their friendship and generosity.

Finally, I am ever grateful to my family. To my parents, Linda, Walter, Norm and Sue, thank you for your continuing support and helping me open new doors. To my brother and his wife, Mark and Janelle, thank you for your encouragement throughout this challenging time.

Chapter 1

Introduction

In this thesis, we present our examination of software tools used for video game creation, specifically those used for developing, or scripting, the narrative and story aspects of video games. Scripting, in this context, deals with creating a varying, often complex, narrative.

We believe, and it is one of the goals of the professional game company with which we worked, that video game (in particular serious game) development should be made accessible and usable to those with low development budgets and little-to-no programming knowledge. Traditionally, varying narratives, or hyper-storylines, have appeared in large, commercial games and are resource-intensive to build. The professional game company we were working with challenged themselves to develop a single game with an hour long game-play in under a month. A far reaching goal was to provide support via tools to enable their customers to create game narrative and content themselves.

The games developed by the company are called 'serious games'. Loosely defined, a serious game is a video game or computer game that is designed primarily to support learning, training or to persuade the user, and not for pure entertainment purposes [93, 91]. Serious game development changes the playing field, with shorter game-play, the addition of learning requirements, the use of varying narrative, and the need for economical development and greater support from tools.

To support serious game development, game scripting tools must be made accessible, with intuitive designs, workflows and presentation of information. In other words, game scripting cannot exist solely in the domain of computer programming and programming languages. We gather evidence that normal procedural programming is not ideal, and that a domain-specific language should be used. To that end, we must understand the process and used tools for game scripting and what needs to

change to achieve the above goal.

We take a human-computer interaction (HCI) focus and therefore utilize frameworks from HCI. Our contributions involve the application and results of employing a variety of human-computer interaction methodologies to study the tools and nature of game scripting, including:

- a survey of existing tools with insights used to create the basis of a pattern language for game scripting and tool design
- an analysis of a successful commercial tool, using a blended form of two inspection methods
- a contextual inquiry into the work of two key members of a professional game development company
- the use of two usability inspection methods to examine an alpha-stage internal tool developed by and for the programmers at the professional game company

The results of this thesis take the form of a model to represent the essential design characteristics of narrative scripts and recommendations for the development of game scripting tools and the details for the next steps in the research and validation.

1.1 Why Study Games?

In general, why study games? We propose three reasons:

1) gamers are everywhere

2) games are big business and

3) games are not just for entertainment

1.1.1 Gamers are everywhere

In many countries, gamers are the majority. For example, according to Pratchett's BBC study entitled *Gamers in the UK*, 59% of 6 to 65 year olds in that country are gamers, with a near-even gender split. Such games are played on consoles, on PCs,

in web browsers, on handheld devices, on handheld gaming systems and on interactive television. The percentages differ by age group, with the following identifying themselves as gamers [83]:

- 100% of 6-to-10 year olds,
- 97% of 11-to-15 year olds,
- 82% of 16-to-24 year olds,
- 65% of 25-to-35 year olds,
- 51% of 36-to-50 year olds, and
- 18% of 51-to-65 year olds.

Similarly, in the United States, approximately 174 million people play games on some sort of platform or device [79, 88].

1.1.2 Games are big business

The numbers presented above mean big business. For example, the gaming industry in South Korea, alone, generated $1 billion USD worth of gaming exports in 2008. In December 2008, the South Korean government spent approximately 240 million USD to fund 60 new gaming projects within their borders to promote even more growth [60]. By the fall of 2008, the combined worldwide sales of the three newest generation gaming consoles (Wii, Xbox 360 and PlayStation 3) were approximately 76 million units [94, 77, 63]. These numbers are just for the hardware console sales, and do not take into account games sold, the numbers of online subscriptions, and sales of extra hardware accessories or paraphernalia. The worldwide sales of three popular handheld gaming systems (Game Boy Advance, Nintendo DS and PlayStation Portable) were at an approximate combined total of 206.7 million units, by fall 2008 [94, 76, 77].

The games market, driven largely by the latest generation consoles and handheld devices, in the US is expected to increase "at a compound annual rate of 6.3 percent to $11.7 billion in 2012 from $8.6 billion in 2007" [84]. For the 25 million Xbox 360 units sold, alone, there are 14 million Xbox Live subscribers. A paid subscription to the Xbox Live service allows players to access downloadable game content, game

upgrades and the ability to play games online, and interact, with other Xbox Live subscribers on their Xbox 360 consoles.

In the world of PC gaming, specifically massive-multi-player online games (MMOG), World of Warcraft reached 10 million subscribers worldwide as of January 2008 [14] and continued to lead the MMOG subscription share, in April 2008, at approximately 62% [109].

1.1.3 Games are not just for entertainment

Pratchett's study reveals that games are not just for fun. Gamers view gaming as part of their identity and would rather game than watch television. Gamers view gaming as both a social activity and solo activity, perceive the gaming market as saturated with certain types of games (like first-person shooting games) and hold a belief that games can and should be designed for other purposes, such as education, persuasion or training [83, 16]. We will examine serious games and four examples of serious games in Chapter 2.

1.2 What is Scripting?

The term 'script' can mean different things, depending on the context. For example, scripts written by playwrights or for movies are much different than scripts in computer programming. They however perform similar purposes: they may be translated or interpreted by someone or something to produce an action or result.

Aldred, et.al., propose 7 types of scripts at work in games, including: dramatic scripts (covering plot, character, dialogue), interface scripts (user actions in game-play such as mouse-clicks), ludic scripts (rules of game-play), interaction scripts (a sum of dramatic, interface and ludic scripts) and implementation scripts (machine code, high-level programming languages, etc.) [2].

While the word 'script' may carry many meanings, and is encountered in a variety of contexts, a distinction of the different types of scripts involved in games is valuable, especially in our attempt to understand what makes scripting a game so difficult.

Conceivably, if a tool could offer support for more technical kinds of scripts and provide effective support developing dramatic and narrative scripts, then perhaps the

barrier to game development would not be as restrictive. Because we are concentrating on varying narratives and stories, we will define and examine interactive narratives and their roles in games in Chapter 2.

1.3 The Plan

This investigation will begin by examining existing literature on games, serious games, interactive narratives and their development. The goal here is to understand the inherent challenges in serious game development. We will concentrate mostly on interactive narratives and the story elements of games. We do not address the development of graphics for serious games. However, we will not exclude looking at interfaces that deal with graphics, maps and setting up the game environments.

Following the background research will be a survey of existing game scripting tools with the goal of identifying key design patterns for game scripting tools.

Next, usability inspection techniques will be utilized while examining the popular, commercial game scripting tool used by Jenkins and other researchers to create serious games. The goal here is to identify salient ideas on the support required for game scripting.

Observations of members of a professional serious game development team in action will follow, including significant time spent observing and interviewing key team members while they work and examining the tools both used by and built by the development teams.

From this research, we hope to form of a model to represent the essential design characteristics of narrative scripts and recommendations for improvement or development of tools to support game scripting.

1.4 Contributions

Much of the work in this thesis has been previously refereed and appears in the literature. In an examination of existing game authoring tools, the basis of a pattern language of design of tools to support game scripting is devised and presented. This work was presented at the European Conference on Pattern Language of Programs

(EuroPLoP) in 2008 and appears in official conference proceedings [56].

We also use the Cognitive Dimensions of Notations framework [43, 44, 45] in combination with an approach similar to Molich and Neilsen's heuristic evaluation [65, 74, 72] to study a commercial game scripting tool. Furthermore, we borrow from other usability techniques to codify and present the observations, representing a small but novel contribution. This approach and our observations were presented at the International Academic Conference on the Future of Game Design and Technology, 2007. The paper also appears in the conference proceedings, published by the Association for Computing Machinery (ACM) [57]. The analysis and conclusions from the paper are expanded upon here in Chapter 4.

In our presentation and the Association for the Advancement of Computing in Education (AACE) paper for the World Conference on E-Learning, we summarize the user-centered evaluation techniques employed in this research, as well as present our use of a contextual inquiry technique in our observations of an Ottawa-area professional serious game development company [55]. The observations, analysis and results of that research are presented in this document.

Most of the material was published in an early form in the literature, and in all cases Marty Kauhanen was the primary author and undertook the research concerned. Throughout this thesis, 'we' refers to Marty Kauhanen under the guidance of his thesis advisor, Dr. Robert Biddle.

In addition to conducting the above research, authoring the above papers and presenting at the aforementioned conferences, Marty Kauhanen also:

- Presented observations to the Ottawa-area gaming company, referred to anonymously as GameCompany (presented in Chapter 5 and 6)
- Aided in the codification and analysis of results in an ethnographic study (see Chapter 6)

All of this work was funded by a research assistance position as part of the a project entitled End-User Scripting for Online Training, funded by a grant from ORNEC (Ontario Research Network for Electronic Commerce).

1.5 Outline

In Chapter 2, we examine serious games, game development, interactive narratives and game scripting. Examples of serious games and game scripting tools are provided, as well as a brief look at a common story structures in games. The concept of a domain-specific language is discussed, as it pertains to a main argument of this work.

In Chapter 3, the basis of a pattern language of design of tools to support game scripting is presented.

In Chapter 4, we describe how we used Cognitive Dimensions of Notations [45, 44, 43] as heuristics in an evaluation of *Neverwinter Nights Aurora Toolset*, a game scripting tool packaged with the successful video game, *Neverwinter Nights*. The analysis and the results of the inspection are presented.

Chapter 5 covers the user-centered observation technique, called contextual inquiry, and how it was used to study a small group at a professional serious game development company.

Chapter 6 involves observations made at the same company, one year later, including a cognitive walkthrough and usability study of their internal development tools.

Conclusions are provided in Chapter 7. Based on the observations, the use of a domain-specific language approach for game scripting is proposed, a model and recommendations are presented, as well as the next steps in the research.

Chapter 2

Background

2.1 Serious Games

As explained in Chapter 1, our objective was to explore scripting for serious games. Serious games have a different set of requirements that change the playing field of game development, including the addition of learning requirements, short overall game-play, a need for decreased cost and development time, while still employing interactive narrative techniques (see section 2.5).

The number and variety of serious games is growing. Sawyer and Smith maintain a taxonomy of serious games [91]. Four illustrative examples of serious games are presented, below.

1) *America's Army*

America's Army is a brand of games, developed by the US Army, as a public-relations and recruitment tool. It is available on multiple platforms, including PC and Xbox 360. The current PC incarnation is called *America's Army: Special Forces (Overmatch)*, version 2.8.4, and the offering for the Xbox 360 is called *America's Army: True Soldiers* [42].

While the game is a multi-player first-person shooter, the purpose of the game is to promote army values and was designed to serve as a recruitment tool. The game website reinforces the message that a career in the armed forces is both viable and honorable, with links to information about U.S. Army careers, interviews with real soldiers and a prominent section dedicated to honoring a "Real Hero" of the US Army [103].

In addition, the *America's Army* platforms have been expanded and utilized the US army and other US government agencies to deliver training content privately within their own organizations [99].

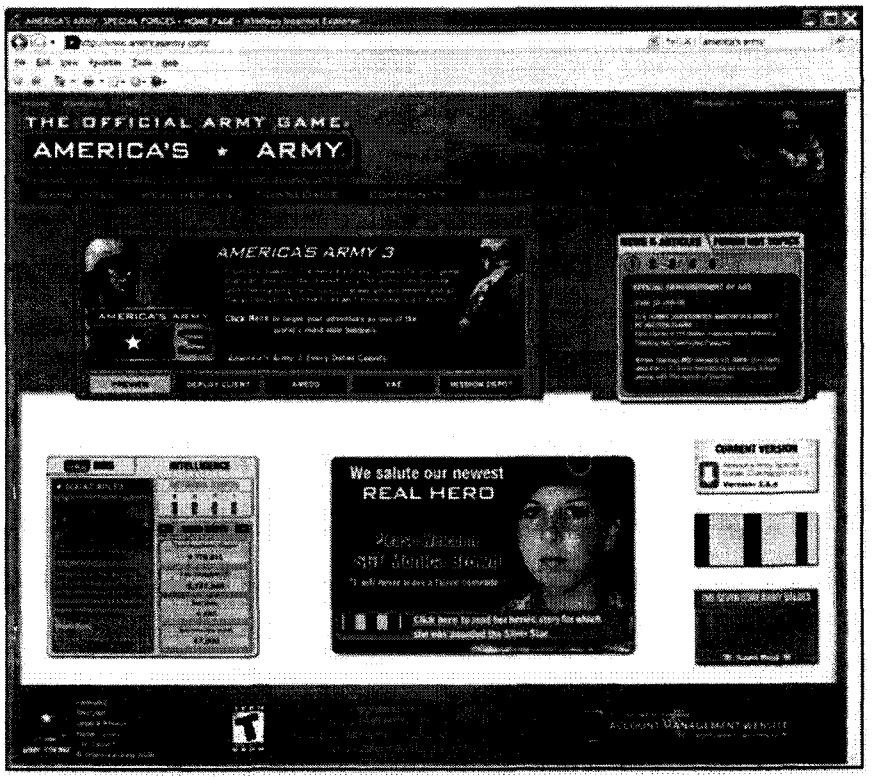

Figure 2.1: Screenshot of *America's Army* website, showing a mix of game-related and real-life content.

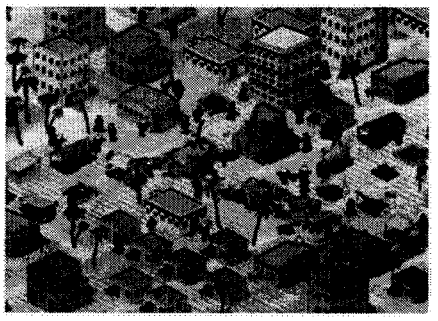

Figure 2.2: A screenshot of *September 12*, an example of a persuasive, serious game

2) *September 12: A Toy World*

America's Army is not alone on the propaganda front. Take *September 12: A Toy World*, for example [39]. This game is best described as a casual Flash-based game that you really only need to play once. The purpose of the game is not fun, but to illustrate through the game-play the following message: bombing (or warfare in general) is not the solution to fight terrorism.

The game-play is simple: you control with your mouse a floating target above some unnamed Middle-Eastern styled city. There are normal-looking citizens bustling about and the occasional dubious-looking terrorist. Clicking a button releases a bomb that has collateral impact on both buildings and non-terrorist citizens. The collateral damage, however, converts citizens into terrorists. Pretty soon, the result is clear: you end up with a rubble-strewn city and a screen full of terrorists! In short, the message is: violence begets violence and hatred begets hatred.

3) *Darfur is Dying*

Darfur is Dying is a Flash-based, serious game that is played in an internet browser. The purpose of *Darfur is Dying* is clear: to raise awareness and elicit sympathy and support and serves as a call to action.

The following statement is found on the game's website [89]:

> *Darfur is Dying* is a viral video game for change that provides a window into the experience of the 2.5 million refugees in the Darfur region of Sudan. Players must keep their refugee camp functioning in the face of possible attack by Janjaweed militias. Players can also learn more about the genocide in Darfur that has taken the lives of 400,000 people, and find

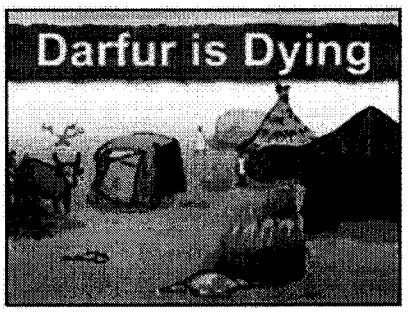

Figure 2.3: The title screen of the serious game, *Darfur is Dying*

ways to get involved to help stop this human rights and humanitarian crisis.

4) *Food Force*

Along similar lines of *Darfur is Dying*, *Food Force* is a game developed by the UN World Food Programme (WFP) to promote awareness about a crisis: world hunger. The *Food Force* website is divided into the following sections:

- The Game: "It's up to you to save and rebuild the island of Sheylan"
- The Reality: "Free Rice: More than just vocab. Find out more about WFP activities and read news from the reality behind Food Force".
- How to Help: "Together, we can make a difference" The game is available for download on both Windows and Macintosh and was designed by a company in Italy and programmed by another group in London.

2.2 Serious Game Development

Serious games occupy a much smaller market share than entertainment games, but serious game market share still has an estimated range of 400 million to 1.5 billion dollars in 2008 in the US alone. These estimates include educational, military, corporate and healthcare spending on serious games. Suggested reasons behind a much smaller share include, the current work-for-hire, or single use, model for most serious game development and high development costs. However, "potentially, every training or education program, every curriculum, could be reshaped by the use of [serious

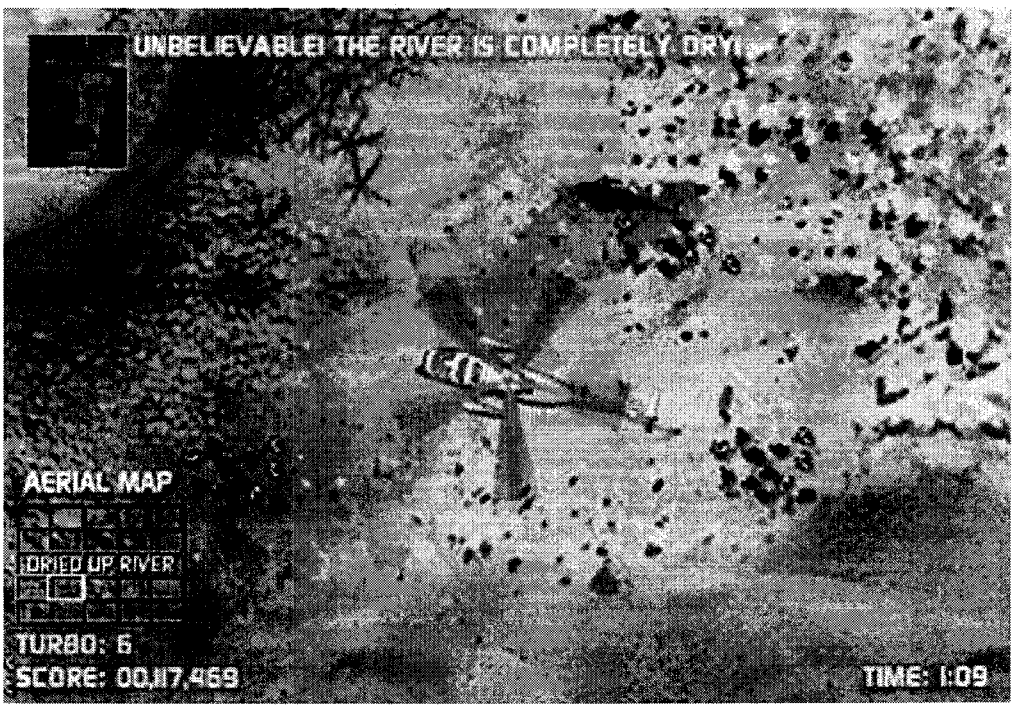

Figure 2.4: Screenshot of game-play in the serious game, *Food Force*

games]" [4, 5]. Aside from *America's Army*, the examples of serious games above are prime examples of single-use games.

Alhadeff [5] offers that a commercial-off-the-shelf (COTS) approach would increase the serious game market value exponentially. This implies that serious games ideally should take the form of other COTS software, like word processors, office products suites and email programs, which are more like "end user tools" in order to lower development costs and reach wider audiences.

Such COTS tools do exist, however. For example, Jenkins and his team at the MIT Comparative Media Studies Program used a COTS video game toolset to build serious games that help teach American history, including one that was set in colonial Williamsburg just before the American Revolution [54]. Jenkins remarked, "It allowed us to develop a game that looked really professional on a low budget and with a small team" [31]. We examine the particular game development toolset utilized by Jenkins in Chapter 4.

It remains unclear that existing COTS software and tools are impeding market growth of serious games. In other words, given that such commercial tools do exist,

then why don't we see a much larger market share for serious games? Sure, an MIT group was able to use a gaming toolset to create serious games, but what about an individual or non-media specialist? Can anyone create a game?

Such questions prompt us to examine the effectiveness of such tools, in their support of game creation, or game scripting, and to examine and understand the practices and challenges of serious game development.

2.3 Why Focus on the Narrative or Story?

Often, games deliver their message in conversations and scenes between the game-play. This is the case with the serious games illustrated above. In *Food Force*, there are cut scenes that deliver information about the status of the fictional island, Sheylan, before and after each of the missions. Before the first mission, aerial surveillance in a helicopter to locate groups of islanders, you're told about the island and its inhabitants and the goals of the mission. The learning about the 'world hunger crisis' happens outside of, or between, the game play. Similar cut scenes are seen in *Darfur is Dying* and mission briefings drive the objectives of game-play in *America's Army*. In *September 12: A Toy World*, the message is quite simple: bombing is not the solution and, while hinted at during in the instructions, is delivered through the relatively simple game play. We will discuss the ideas of ludic vs. narrative perspectives and procedural rhetoric in Section 2.6.

Concentrating on the narrative and story aspects of games then brings us to the realm of interactive fiction (IF) and interactive drama (ID), which we will collectively call interactive narrative (IN), defined below. Aarseth notes that "despite lavish and quite expensive graphics of these productions, the player's creative options are still as primitive as they were in 1976" [1]. One might suspect that one of the barriers to entry for IN development is technical know-how.

Anecdotal evidence from conversations with teachers at E-Learn 2007 conference revealed that most teachers understood the value and desired to deliver and support their cirrucula via games. Most teachers, however, did not have the technical background, wherewithal, and time to invest to do so.

2.4 Interactive Narratives

Interactive fiction (IF) is a literary narrative genre often employed in video games. IF systems accept textual commands from the user, traditionally in the form of verb-noun pairs or natural language statements, to influence the environment and determine the outcome of the story or game. While IF systems have an interactive system that is restrictive to text, their study may be illumitating for video game design and development. Many video games employ an IF system within them. Interactive fiction is goal-oriented. Historically, interactive fiction was heavily influenced by *Dungeons and Dragons* [52, 46] and were usually adventure-themed [81] with strong elements of exploration, mystery and puzzle-solving. Others interactive fiction systems, such as *Graphic StoryWriter* [96], were created as a story environment for early readers and thus were influenced by children's stories.

Interactive drama (ID) is quite similar to interactive fiction in that it refers to a type of drama where the audience plays an active role, modifying the drama by changing the course of actions [98]. A contemporary example of an interactive drama is *Façade* [86], a first-person, 3-D virtual world that allows the player to interact with computer-controlled characters. Interactive drama is experience-oriented. For the purpose of this thesis, we shall include both interactive fiction and interactive drama under the umbrella term: interactive narrative (or interactive storytelling). A common characteristic of interactive narratives is that they have multiple, and often innumerable, outcomes. The interaction gives the player a sense of agency and the innumerability of the outcomes of the games promotes replay-ability.

While entering verb-noun pairs (i.e. Go West) into a command line prompt is perhaps an outdated interaction, Aarseth argues that the "ergodic structures invented by Crowther and Woods twenty years ago are of course far from dead but instead persevere as the basic figure for the large and growing entertainment called [...] 'interactive games' [and] despite lavish and quite expensive graphics of these productions, the player's creative options are still as primitive as they were in 1976" [1]. 'Ergodic' with respect to text means that it takes the author less than trivial effort to traverse, more effort than skimming and flipping pages.

In general, "when game-literate players encounter options in dialogue, they feel

obligated to listen to everything that particular [non-player] character says" or they might miss some crucial information [8]. This type of interaction is most suitable for adventure games and computer role-playing games.

2.5 Ludic vs. Narrative Perspective in Games

The two main perspectives on how meaning is best constructed in games are: the ludic and narrative perspective. The ludic perspective suggests that the player constructs meaning from the game play itself and not just from the narrative. Bogost provides the example of how children learn about mortgages, work ethic, managing money and debt in the game *Animal Crossing* through the rules of the game and the game play itself. Bogost calls this "procedural rhetoric" [16, 17]. While this type of game play can make a powerful impact on the user, the main criticism against the ludic perspective is that it is really poor in imparting any context or background for a story. Narrative is still important.

In the four examples of serious games, above, we see that they employ a combination of narrative and ludic structures to impart meaning, to varying degrees. *America's Army* has a whole suite of training missions which teach the player about military conduct, equipment and machinery. It is both learn-by-doing and instructional in nature. Mission briefings are used to provide context for a mission. In both *Darfur is Dying* and *Food Force*, the instructions are crucial to completing the game tasks and the knowledge about the various crises is imparted in narrative form. The game-play action of 'flying around in a helicopter looking for people' would hold little meaning in *Food Force* without the accompanying narrative context. In *September 12: A Toy World*, the only possible action in the game-play results in destruction, death and waves of new terrorists. This game, of the four, is strongest in its procedural rhetoric. It does however contain instructions at the start of the gameplay that, in combination with the game-play, deliver a one-two punch in the message.

2.6 Story Structure in Games

In this subsection, we examine common story structures in video games. These story structures also lend themselves to narrative structures, where dialogue options are branches. The structures themselves illustrate the complexity of managing the creation interactive narratives in software.

2.6.1 Gating the Story

Gating is a method of mapping a linear story to a nonlinear game. It involves a plot unfolding as a sequence of linear checkpoints, each unlocked by the player's actions when faced with challenges or explorations in a section of the game. For example, the mapping could be a story event, such as a cut scene, followed by 3 unordered challenges that result in a trigger when they completed. The trigger in turn starts another story event, such as a crucial plot-related conversation, that may lead to more unordered challenges [8]. This is a common method that is found in many commercial games, including computer role-playing games like *Neverwinter Nights* [10].

2.6.2 Branching Story Structure

A branching story structure is one that allows options in dialogue and plot, to be selected by the player. At each selection point, the plot branches. This type of structure, however, leads to "a combinatorial explosion of unused resources — 3D environments, character animations, dialog[ue] that the players never experience" [8]. It does lead to multiple endings, arguably promoting replay-ability. One way to mitigate this is to have some of the branches recombine.

2.6.3 Branching Story Structure, Recombining Branches

Another branching story structure is one with paths that separate and recombine. In effect, there are parallel storylines that both end in the same plot point. *Deus Ex* makes use of this type of structure [51, 8]. Game stories with branching and recombining may or may not have multiple endings [41]. *Knights of the Old Republic*

is an example of a game that uses this branching and recombining, has multiple endings and employs gating [11].

2.6.4 Other branching structures

Other branching structures include a grid branching structure, where players may move from one intersection on a grid to another, and a complete graph, where all nodes are connected and a player may move from one node to any other node [41].

2.6.5 Hypertext

A hypertext document contains links to parts of the same document or to other texts and is the foundation of the World Wide Web [71]. Glassner suggests that the danger of using hypertext solely as a means of structuring a story is that it can destroy the experience of linearity and is best suited for referencing information and some poetic expression [41].

2.6.6 Opinions in Literature

While Glassner argues that branching narratives and hypertext narratives rip the audience out of the story, he does admit that they are "overwhelmingly the two most popular forms of giving audiences a say in the development of a story" [41]. The main reason for the his harsh view involves the "fundamental conflict between plot and interactivity" [26].

Crawford mitigates this known conflict by looking at metaplot (driven by rules) as opposed to the plot (driven by events) and how metaplot does not have conflict with interactivity. He also espouses giving players interesting choices [26]. Whereas Glassner awaits advances in "the technology to understand free-form language and craft a reasonable reply" but he acknowledges that it "simply doesn't exist yet" [41].

2.6.7 Autonomous Characters

While toolkits for creating interactive narratives and video games do exist, Glassner suggests this area needs work: "The largest missing piece of technology in creating

story environments and the characters that inhabit them, is the software required to create and control them. So instead, everything is pre-scripted". The missing parts are a "customizable personality system for creating characters", "a robust way to deal with language", and "some kind of autonomous characters that people will find sufficiently interesting and adaptive" [41].

Chris Crawford has been working on interactive storytelling and autonomous characters for the better part of seventeen years and is the inventor of *Erasmatron* [24, 25] and its next incarnation, *Storytron*. The non-player characters in the games "are intelligent, and they each have their own personalities and agendas. They respond to your choices in emotionally complex and challenging ways, so you must consider the impact of your choices on your relationships with them" [23].

Storytron has an authoring tool, called *Storyworld Authoring Tool* or *SWAT*. The intelligent and complex nature of the non-player characters result in a lot of content to be created. For example, *SWAT* requires authors to break everything down into very small elements, using a subset of English, to portray the possible actions of the story. This is done in a visual language called Deitko [23, 85] (see Figure 2.5). The main criticisms are *SWAT*'s difficulty to learn and use, in part by the requirement of learning a new language to represent stories and by the complex visualizations that represent the story [6, 85].

Due to this high complexity, of both game play and creation, and the fact that only a subset of language is used in the system (leaning more towards ludic elements than narrative), we will not consider autonomous characters any further.

2.7 Domain Specific Language

2.7.1 Definition

A domain-specific language is one that facilitates the solving of a problem or building a model in a specific problem domain, as opposed to a general purpose programming language, like *C#* or *Java* [30, 95].

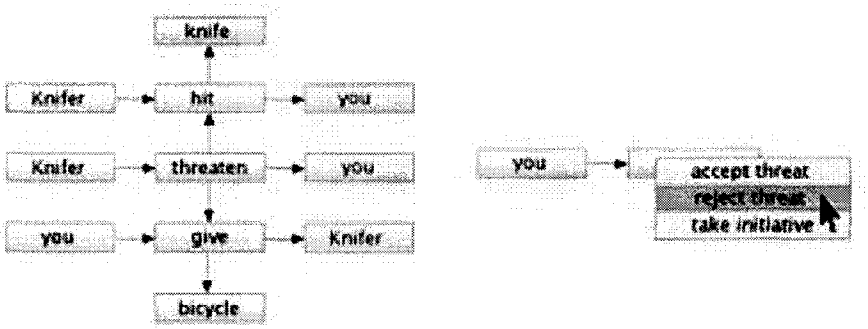

Figure 2.5: Example of Deikto representation of a mugger threatening the user with a knife (right) and the user's possible actions (left) [23, 85]

2.7.2 Advantages

Reasons for using and building domain-specific languages include [87]:

1. making a technical task easier
2. expressing rules and actions in a manner that is closer to a domain, so that non-programmers may understand them
3. automating actions and tasks
4. exposing the internal workings of a program to the outside world

2.7.3 Example: *LabVIEW*

LabVIEW, by National Instruments [68, 67], is a prime example of a graphical programming environment that employs a domain specific language. It leverages a paradigm specific to building test, instrumentation, measurement and control applications and the expertise of its users: circuit diagrams. In other words, *LabVIEW* is primarily used to create programs that control and monitor electronic equipment and the programming language looks like circuit diagrams.

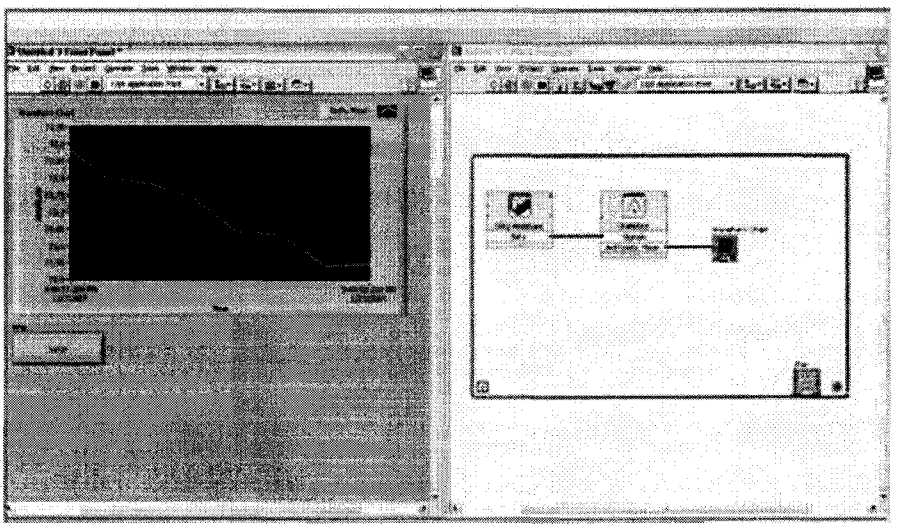

Figure 2.6: Screenshot of *LabVIEW* from online tutorial [68]

The UI is divided into two sides: one for building the UI of the program and another for the block diagrams that define the I/O of the program, data sources, functionality and data processing. The block diagrams are drawn using blocks and wires, with the data flow line indicating the sequence of execution.

The UI building pane follows the WYSIWYG (What You See Is What You Get) approach and is executable. The example in Figure 2.6 displays data being gathered from the inputs (left) as defined in the block diagram (right). The block diagram contains a graphical representation of the modules that are required to interface with the hardware and generate and display the data. The *LabVIEW* programmer neither needs to write hardware drivers nor know programming languages. The *LabVIEW* language follows the conventions of circuit diagrams to assist engineers while working in their domain.

2.8 Petri Nets and the PN-model

Concurrent to the research performed in this thesis was the latest work involving modeling stories and narratives for games using Petri Nets and variants of Petri Nets.

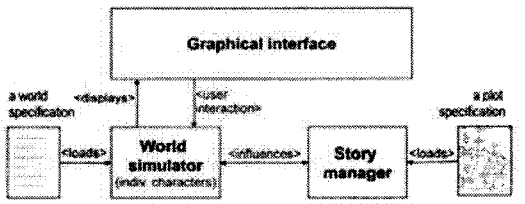

Fig. 1. Two-layered control architecture of a "typical" story-telling application. Simulator of the virtual world features individual characters and controls their background behaviour. This behaviour is modified for the narrative purpose by the story manager. Each component uses its own representation.

Figure 2.7: Figure 1 from *Hierarchical Petri Nets for Story Plots Featuring Virtual Humans* [7]

2.8.1 Petri Nets

Petri Nets are a form of specification used in software engineering and have been used to specify behavior of characters [13], plot monitoring in simple games [29], for prototyping games [21] and representing plots in a serious game that was a more of a simulation with no characters [18].

Like deterministic finite state machines (dFSM), Petri Nets are formal, exact and graphical. Both Petri Nets and dFSM allow for branching stories. The main advantage of Petri Nets over dFSMs are that they allow for expressing parallel game states, while dFSMs do not. Petri Nets also are better suited for very large virtual worlds with a hundred virtual characters or more [21, 18].

2.8.2 PN-model

Balas takes into account the need for an organized story structure and a need to divide high-level story logic from lower-level rules. He proposes a custom variant of Petri Nets to model and storytelling applications, called the PN-model [7]. This variant is hierarchical and is used to define plots to be read by their PN-model engine and they developed an interface between the engine and the virtual world simulator, "through which the story manager influences the world and vice versa" (see Figure 2.7). Like Petri Nets, the PN-model allows for parallel storylines. The advantage over Petri Nets is that it allows for a hierarchical decomposition of stories. In the PN-model

the story manager uses triggers to monitor the state of the world, through the use of tokens and variables, can access virtual world objects. In addition, the expressive power of the triggers allows for logical operations (And, Or, Not) as well as first-order logic (All, Exists, Foreach), allowing for complicated expressions [7].

2.8.3 Do we need Petri Nets and PN-Models?

Brom admits that if one is writing a story with a preset plotlines, that "do not evolve in parallel, deterministic finite state machines would likely be sufficient" [18]. The creator of the PN-model admits that "understanding and implementing [PN models] can be demanding" [7] and their design process involves at least two people: a narrative writer and a programmer that is converts plot specifications manually into code. An authoring tool has been labeled as 'future work'.

In other words, without an authoring tool that provides support to the narrative writer, the expertise of a programmer is still required to incorporate the narrative writers' content into code. In addition, both Petri Nets and PN-model are suited to large virtual worlds and complex, parallel game plotlines. Support for very large virtual worlds and parallel plotlines were not seen as a requirements for the serious games with an hour long game-play, as developed by GameCompany.

2.9 Conclusion

The study of interactive narrative components of game scripting is relevant, as dialogue in games is prevalent and IN provides a means of presenting ergodic structures, and narrative structures are just as important as ludic structures. Narratives in both *Darfur is Dying* and *Food Force* play a crucial role in framing the action scenes and delivering the message and learning content.

We examined the common story structures in video games, as managing their complexity is part of game creation.

While we highlighted the latest in interactive narrative research, including autonomous characters and the use of Petri Nets for modelling stories, we believe these introduce a level of complexity that may not be warranted given our goals of supporting interactive narrative scripting with the non-programmer in mind and the shorter

durations of most serious games.

We illustrated a prime example of a successful commerical tool that employs a visual domain specific language to support its users. We will keep this approach in mind as we examine game scripting.

In the next four chapters, we examine how game scripting, with a focus on interactive narrative, is supported in contemporary toolkits, how they should work from the user's perspective and how narrative scripting actually plays out in practice.

Chapter 3

Patterns for Game Scripting Tools

3.1 Introduction

Interactive narratives are an important element in entertainment games and serious games alike. Their scripting tools, however, have their roots in programming languages and programming environments. These tools provide a means for users to create custom content, but they are not all necessarily intuitive or easy to use for non-programmers.

In this section, we review the literature and examine the tools and techniques for interactive story-telling and we identify common design patterns [3, 40] to support the scripting and content creation for computer games, specifically interactive narratives. We have organized these patterns in the form of a pattern language to clarify how they relate to support of the design of interactive narrative software systems, in particular their end-user development (EUD) tools and environments [97] or, simply, scripting tools.

We have identified patterns that suggest that instead of relying upon the language, structure, semantics and terminology of other domains, like programming, one should instead consider the language and concepts from the domain of story-telling when designing scripting tools for interactive narratives. The patterns are derived, with examples, from existing game scripting tools. These tools include IN tools like *Adrift*, *Inform 6* and *Inform 7*, as well as end-user game development tools that also include IN systems, such as the *Neverwinter Nights Aurora Toolset*. In other words, there are some common elements amongst the tools and examples of designs that constitute an important domain-specific language (see Chapter 2). We also draw upon examples from end-user development tools specific to other domains, like *LabVIEW*, and programming environments.

In toto, this section provides the basis of a pattern language of design for authoring tools for story-tellers and interactive narrative writers who are not programmers. Because narratives are important to serious games, these patterns may also apply to the support for serious game scripting, in particular for non-programmers. Moreover, serious games are more likely to contain input and content from subject matter experts in small organizations as opposed to professional writers at game development companies.

3.2 Basis of a Pattern Language

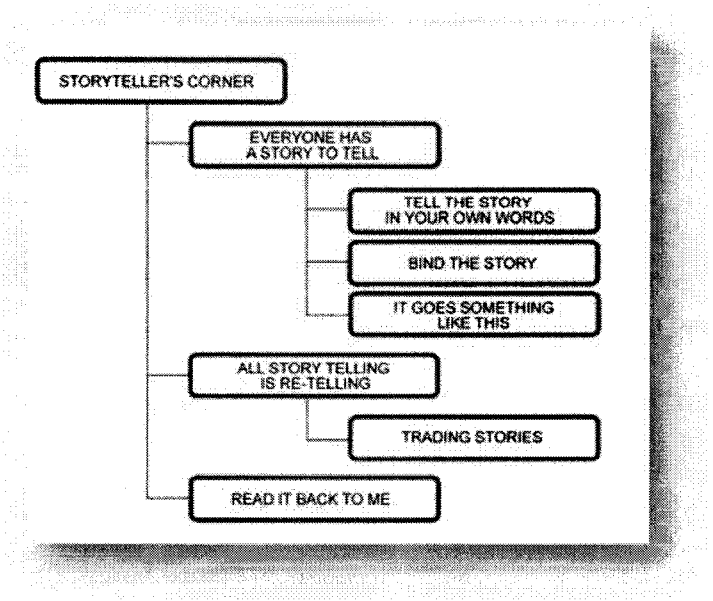

Figure 3.1: The basis of a pattern language for design of end-user development environments for scripting interactive narratives

The individual patterns may be summarized as follows:

STORYTELLER'S CORNER: Provide an authoring tool for game environments.

EVERYONE HAS A STORY TO TELL: Authoring tools should support non-programmers.

TELL THE STORY IN YOUR OWN WORDS: Using natural language as a programming language will be more accessible to creative writers.

BIND THE STORY: Using story metaphors in the organization of UI layout, data and concepts in the authoring tool will create a familiar structure for creative writers.

IT GOES SOMETHING LIKE THIS: Using domain-specific terminology in the authoring tool will also support creative writers.

ALL STORY TELLING IS RE-TELLING: Provide a library of resources to support re-use of story assets, characters and story elements.

TRADING STORIES: Provide a means of sharing story assets with others and between stories.

READ IT BACK TO ME: Provide a means to play-test the story.

The patterns also are related in a hierarchical way, as shown in Figure 3.1. STORYTELLER'S CORNER is the top level in the pattern language for design of end-user authoring tools for interactive narratives. It proposes a specific environment where storytellers will create their own stories and content in an interactive narrative system. Within the STORYTELLER'S CORNER, authors should: (a) be comfortable and not impeded by the authoring environment, (b) be supported with a library of story resources, and (c) be able have their stories read back to them, even if they are not complete. These three issues are addressed in EVERYONE HAS A STORY TO TELL, ALL STORYTELLING IS RETELLING and READ IT BACK TO ME, respectively.

EVERYONE HAS A STORY TO TELL is broken down into three sub-patterns: TELL THE STORY IN YOUR OWN WORDS, involving the use of natural language, BIND THE STORY, involving the use of metaphor and IT GOES SOMETHING LIKE THIS, involving the use of the terminology of stories over the terminology of programming within the authoring environment. In other words, these three patterns are specific ways in which authoring tools can provide support to creative writers.

ALL STORYTELLING IS RETELLING is supported by the sub-pattern, TRADING STORIES, allowing authors to share their custom resources.

3.3 Alexandrian Pattern Form

The design patterns here follow the Alexandrian pattern language format [3]. A pattern in this form follows this basic structure: a problem headline, introduction, problem statement (bold), solution statement (bold), explanation, consequences, illustrations and relationships with other patterns.

3.4 The Patterns

3.4.1 STORYTELLER'S CORNER

One of the challenges for video game developers is fostering a culture of commitment for the game. Another challenge is creating scale and variety within the game sufficient to justify such devotion. Both these challenges can be met by providing the means for customers to create their own custom content. Many games that have adopted this strategy have provided authoring tools or the ability for customers to create mods, or modifications, as well as new game content, resulting in active and creative online communities. For example, popular strategy video games with modification capabilities include *Civilization 4* and *Age of Empires* [36, 35].

The success of a video game involves player commitment and variety on a large scale.

Therefore: Provide the tools necessary to create custom content for games.

There are many interactive narrative systems that provide the tools for the creation of custom content. *Neverwinter Nights* is one such example. As of May 2005, there were over 4000 community created game modules for the *Neverwinter Nights* game, downloaded over 250 000 times [80] from the *Neverwinter Nights Vault* web site (http://nwvault.ign.com/). This community site continues to act as a repository for custom content for both *Neverwinter Nights* and *Neverwinter Nights 2* games.

The consequence of providing story-authoring tools is the ability for users to create their own stories. Without this, the user becomes reliant upon the developers for new content. *Façade* (a first-person, real-time, AI-driven, non-linear, generative, 3-D interactive drama) is cutting edge but, at the moment, lacks an authoring tool. With

over half a million downloads since July, 2005, *Façade* unfortunately boasts only one act [86]. While the act may be replayed over and over with different results, it remains limited in its appeal because there is no means of creating new interactive narratives.

As illustrated in TELL THE STORY IN YOUR OWN WORDS, the technology behind *Façade* is complex. No doubt the complexity of creating and adding an authoring tool would be compounded by the complexities of the *Façade* architecture itself.

Adding a STORYTELLER'S CORNER to a game system raises two areas of concern for programmers. First, the addition of authoring tools adds complexity and difficulty to the task of programming the entire system. For example, the game engine cannot be static; it must dynamic, capable of accepting a variety of different inputs as determined by the user. The second area of concern is that a story authoring environment must also be easy to use. The sub-patterns of STORYTELLER'S CORNER attempt to mitigate the second concern to make it easier to use, but leave the handling of the dynamic textual input to the domain of programming.

3.4.2 EVERYONE HAS A STORY TO TELL

An interactive narrative system is a piece of software, and the act of scripting a story is similar to writing a program. Indeed, game systems are software and are created by game developers by programming. But game players and potential authors of new content are not always programmers, nor do they need to be in order to add content at the level of the story. While interactive narrative software systems require some technical knowledge, most story-writers and authors are not programmers. This may make it difficult for authors to tell a story.

It is difficult for story-writers to work in programming environments.

Therefore: Make the story-authoring tool accessible to writers. EVERYONE HAS A STORY TO TELL **and anyone who can tell a story should be able to script one at the higher level of story-telling, rather than the lower level of programming.**

While story-writing and programming are not mutually exclusive, it does not make sense to burden writers with learning to program in order to create interactive narratives. The solution is to provide a higher level representation of the programming.

As discussed in the parent pattern, the increased complexity of programming the system and the tool itself is a tradeoff (and consequence) for an increase in the tool's ease of use. This is a natural progression in technology, however. Few people still program in assembly code, a low-level language that deals directly with memory addresses, registers and stacks, but choose to program in more abstract, high-level languages, such as *C++*, *C#* or *Java* that deal in terms of Boolean expressions, classes, functions and variables.

The *Neverwinter Nights Aurora Toolset*, for example, is an even higher-level language still. It is very graphical in nature, allowing authors to drag objects, such as monsters, buildings and trees, onto an area map. The objects are editable through property screens that consist of fields for parameters.

The *Adrift Generator*, the authoring tool for the *Adrift* text adventure software, takes a similar approach with respect to programming [107]:

> Instead of having to learn a new adventure programming language, ADRIFT Generator takes all the difficulty away leaving you with a simple, yet powerful game designer. Adventures are built up by adding rooms, objects, tasks, events and characters. All you have to do is type in the descriptions, and select how everything interacts with each other from pull down menus and lists.

The creators of *Façade* realize the advantage of a higher level language for authoring, stating [86]:

> Our current tools and authoring techniques require programming expertise; in fact, even expert programmers familiar with languages such as C++ and Java will have to learn new ways of thinking about programming to program in ABL. For this reason, we are not publicly releasing the authoring tools at this time, though we will be working towards the future release of higher-level authoring tools that enable writers and artists to create Façade-like content.

Their custom language, A Behaviour Language (ABL), is a behaviour language

that supports sequential, parallel and joint behaviours and is "effectively a multi-threaded programming language [and] challenging to program in, even for experienced coders" [61]. The designers of *Façade* realize that, due to the complexity of their system, to create a STORYTELLER'S CORNER would require them to follow the pattern EVERYONE HAS A STORY TO TELL. There are several sub-patterns that one may employ to create a tool that limits the amount of programming knowledge required to script a story. The three sub-patterns of EVERYONE HAS A STORY TO TELL are: TELL THE STORY IN YOUR OWN WORDS, BIND THE STORY and IT GOES SOMETHING LIKE THIS.

3.4.3 TELL THE STORY IN YOUR OWN WORDS

Programming languages are used to write software. This means that for most video game systems and interactive narrative systems, the ability to create custom content is tied to the author's ability to program. This is especially true if a video game system does not provide an authoring tool. STORYTELLER'S CORNER, however, proposes a specific authoring tool. EVERYONE HAS A STORY TO TELL tells us to reduce technical knowledge requirements by giving the author a higher-level environment to work in. But how do you provide a way to script the logic of a story and the story itself that is both easy to read yet sufficiently powerful?

The narrative structure necessary for interactive story-telling requires management of sequences, conditions, repetition, and remembering details that happen along the way. Programming languages can do all this but they are difficult to learn and difficult to read. In the end, the program's code neither reads nor looks like the story that it represents. Poor readability results in a longer time to learn the system, and problems with the maintainability of the code. In other words, an ordinary programming language will not reveal the author's intent.

Creative writers have difficulty reading, writing and translating stories into code in a programming language.

Therefore: Use a limited natural language as a programming language.

Instead of relying upon programming languages, a story-authoring system could adopt rules for semantics and English sentences making it much closer to the structure

```
Object  cloak "velvet cloak"
  with  name 'handsome' 'dark' 'black' 'velvet' 'satin' 'cloak',
        description
           "A handsome cloak, of velvet trimmed with satin, and slightly
            spattered with raindrops. Its blackness is so deep that it
            almost seems to suck light from the room.",
        before [;
            Drop, PutOn:
                if (location == cloakroom) {
                    give bar light;
                    if (action == ##PutOn && self has general) {
                        give self ~general;
                        score++;
                        }
                    }
                else
                   "This isn't the best place to leave a smart cloak
                    lying around.";
            ],
        after [;
            Take: give bar ~light;
            ],
  has   clothing general;
```

Figure 3.2: A code snippet from the *Inform 6* version of *Cloak of Darkness*, describing the behaviour of the velvet cloak object. This code is from the middle of the file.

of work in creating stories. Instead of objects, classes and methods, the author could concentrate on writing more natural sentences and dialogue. Natural language is used to script the story itself. Of course, the issues that must be handled will be similar to those in programming, but use of more familiar syntax and terminology will assist creative writers.

We see the evolution from object-oriented code to natural language-based code in the *Inform* [53] interactive fiction design system. In *Inform 6*, the language used in story creation is essentially an object-oriented programming language that is compiled to a file which is then interpreted in the user's machine. In Figure 3.2 and Figure 3.3 are two code snippets of *Inform 6* code from the 'Hello World' of interactive fiction stories called: *Cloak of Darkness* [37].

Notice that the above code does not look anything like a story and reads very poorly. The "Object cloak" is the class definition of a cloak that describes the behaviours of a cloak object the game and the "Initialize" portion, in the second snippet,

```
[ Initialise;
    location = foyer;
    move cloak to player;
    give cloak worn;
   "^^Hurrying through the rainswept November night, you're glad to see the
    bright lights of the Opera House. It's surprising that there aren't more
    people about but, hey, what do you expect in a cheap demo game...?^^";
    ];
```

Figure 3.3: A second code snippet from *Cloak of Darkness*, written in *Inform 6*. This snippet is located near the end of the file and is the code that instantiates the game-play.

is the part of the text that the player sees at the beginning of the interactive narrative. Clearly, one must understand classes, methods, instances, if-then-else statements and variables to understand (and write) the above code.

In *Inform 7*, the objects, class and methods are replaced by a rule-oriented design and natural language where "the activity of programming IF is a form of dialogue between programmer and computer to reach a state with which both are content, [...] not unlike the activity of playing IF" [70].

Daly summarizes the advantages and disadvantages of using natural language in *Inform 7* [28]. Advantages include: the ease of reading, the elimination of interaction fiction-specific errors, the limiting of loop structures and the lack of programming concepts like procedure names and variables. Daly notes that the disadvantages of using natural language "include occasional awkwardness of natural language and the possible penchant for writers to misunderstand the meaning of a paragraph of code when skimming" [28]. Also, when writing natural language-based rules, one must consider the alternate verbs that a player may use to perform an action. For example, see the rule as it appears in the *Inform 7 Cloak of Darkness* in Figure 3.5.

Basically people put on cloaks to wear them, but hang cloaks on brass hooks. This potentially could be time consuming to write rules for all the alternatives, but consider the *Inform 6* version of the above rule (in Figure 3.6).

While some diehard programmers may consider natural language programming too verbose and less powerful than an object-oriented oriented approach, they would

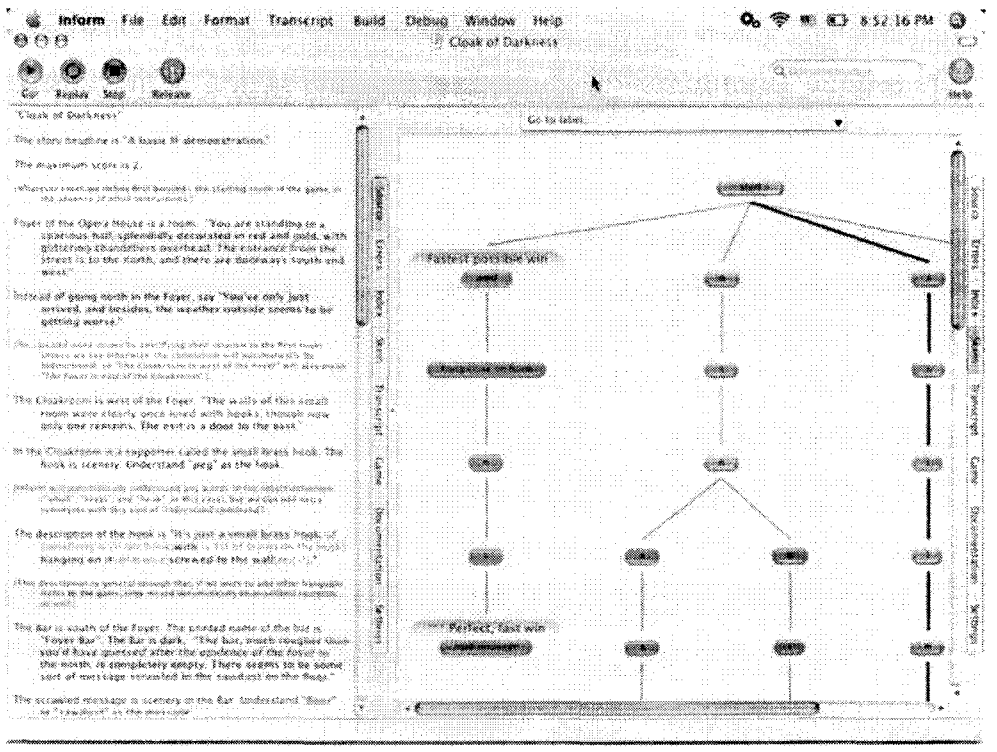

Figure 3.4: A screenshot of *Inform 7* showing the natural language 'source code' of the *Cloak of Darkness* [37]

```
Understand "hang [something held] on [something]" as putting it on.
```

Figure 3.5: Rule Example in *Inform7* from *Cloak of Darkness* [37]

```
Include "Grammar";

Verb 'hang'      * held 'on' noun     -> PutOn;
```

Figure 3.6: Rule Example in *Inform6* from *Cloak of Darkness* [37]

be happy to know that *Inform 7* supports the ability use *Inform 6* code in its projects [28].

TELL THE STORY IN YOUR OWN WORDS is one of three sub-patterns in EVERYONE HAS A STORY TO TELL. The other two, possibly complementary, patterns in design are: BIND THE STORY and IT GOES SOMETHING LIKE THIS.

3.4.4 BIND THE STORY

Discussed in EVERYONE HAS A STORY TO TELL and TELL THE STORY IN YOUR OWN WORDS was the need for a higher-level representation of the code and how natural language may be used in lieu of programming languages, respectively. The organization of the code and environment is addressed in BIND THE STORY.

While the work of creating an interactive narrative is similar to that of writing a program, the motivations and criteria for success have a different emphasis than for programs. An interactive narrative has a structure, often branching, and the writer will often want to read through the various story paths of the IN. In particular, it is desirable for each story path to have the proper pacing, flow and stylistic consistency. Remember that a story, as experienced by the player, is simply one of the paths through the structure of the interactive narrative.

Just as the language of the story should be more like a story and less like a program, so too must the STORYTELLER'S CORNER support the process followed by the content author in a way more suitable to the work and space of story-telling. The visibility and understandability of the environment and story language within are important.

The environment for creating interactive narratives is typically organized like a programming environment, but the users are typically creative writers and not programmers.

Therefore: Employ story metaphors to help with the structure of work in story design and the organization of story elements in the authoring tool.

By leveraging concepts in the domain of stories and drama, the creation of interactive narratives can be less arduous and more intuitive for the author. In the

interface of *Inform 7*, Nelson intentionally uses a book metaphor, where a project is a single book and the file system view of resources has been removed entirely [70]. In Figure 3.4, we see a book with the source code on the left, written in a natural language divided into paragraphs, with the results of the natural language code displayed on the right. The tree diagram illustrates the overall structure of the narrative and one path through the structure is a story as experienced by the player. Few programming concepts are required to understand the interface. The paragraphs provide a story-like way of breaking up the natural language code.

The use of metaphor can be seen in end-user development environments for other domains. The canonical example is *LabVIEW*, a diagram-based programming environment used to create control programs for electronic equipment. *LabVIEW* employs a wiring metaphor where block diagrams are the basis of the source code, leveraging the skills already possessed by engineers and scientists in that domain [67]. Another example of metaphor use is found in a software system that uses robotic agents on virtual tracks to program the behaviour of actual robots [22]. In addition to metaphor, these end-user environments are employing visual programming techniques, a much higher-level representation of the underlying code.

A conceivable disadvantage is that the use of metaphor in a design tool may seem contrived. For example, the designers of *Inform 7* notably did not rename the menu item 'Debug', a technical term, as something that fit perfectly with the book metaphor. Debugging is the act of finding and correcting the errors, or bugs, in a program. Debugging is different than reading through a story to see if it makes sense or even play-testing an interactive story (the Run command in *Inform 7*). A debugging tool finds errors in logic and syntax automatically and informs the user of the errors. Implicitly, the designers of *Inform 7* say: adopt aspects of other metaphors when such aspects are needed but cannot be accurately portrayed in the main storybook metaphor.

3.4.5 IT GOES SOMETHING LIKE THIS

Often the terminology used in an authoring environment is borrowed from programming environments. Would-be authors of interactive narratives may not be

familiar with technical terms, error codes and programming commands. The intent of providing the terms and codes is to help and, in order to help, they must be understandable.

The supporting environment for creation of interactive narrative content is similar in nature to that for supporting software development in that one manages static 'text' that is stored and later used to produce dynamic behaviour. But the vocabulary for managing all this text has grown up within software development is unfamiliar to story-tellers.

Environments provided to create interactive narratives contain terminology from programming, but the users are non-programmers and do not understand the terminology.

Therefore: In the authoring tool, use terminology that either relates to story creation or that is less technical in nature.

By incorporating terminology from the domain of the stories and drama, the creation of interactive narratives may be more intuitive for the author. The author will be more inclined to use commands that sound less technically intimidating, and will be more likely to understand the error messages.

The use of technical terms and commands is indicative of the evolution of authoring tools from programming environments to higher-level story-creation environments, as previously illustrated with *Inform 6* and *Inform 7*. While *Inform 7* does keep the term 'Debug' as it doesn't seem to have a story-related equivalent, it does minimize the technical nature of error messages, themselves. *Inform 7* provides a different type of error message, called a 'problem', that is not like a traditional programming compiler error. Problems make reference to sections and chapters, not lines numbers, and make liberal use of quotations in order to make suggestions as why the problem occurred. In addition, the compile-run command in *Inform 7* is labeled simply 'Go', a much more friendly term to a non-programmer [70].

In the *Façade* architecture (in addition to the ABL language, an animation engine and an interpreter for natural language input) there is a 'drama manager' that "sequences dramatic beats" [61] where beats are a basic unit of measurement in a story. The use of beats is both a use of IT GOES SOMETHING LIKE THIS, dealing

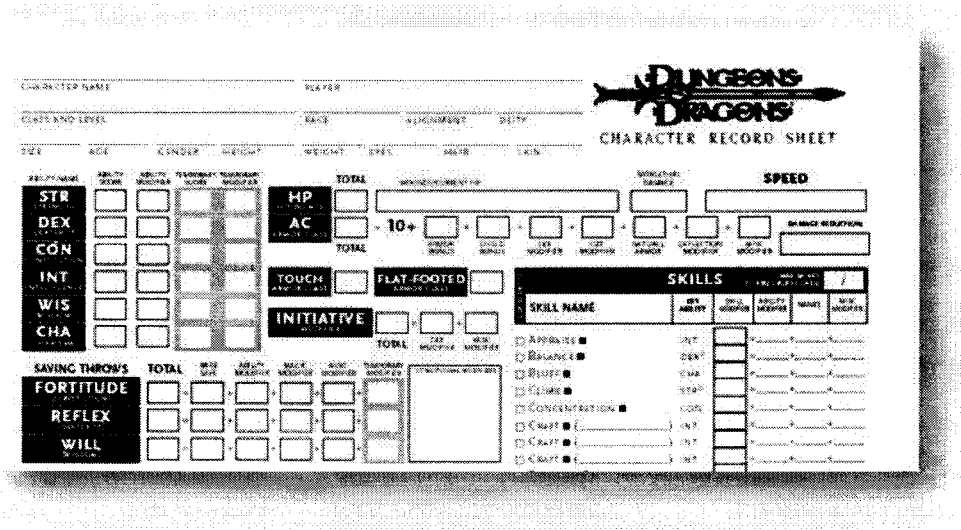

Figure 3.7: The top portion of the first page of a pen-and-paper *D&D* Character Sheet [102]

with beats as a unit of measurement, and also an example of BIND THE STORY, as it adopts the beat as a drama metaphor.

In the *Adrift* architecture [107], the *Adrift Generator* and the *Adrift Runner* will appear to a programmer like a compiler and interpreter, respectively. 'Generator' and 'runner' are much more understandable and operational terms for non-programmers.

An example of a system that employs all three sub-patterns of EVERYONE HAS A STORY TO TELL is the *Neverwinter Nights Aurora Toolset.* As previously mentioned, *Neverwinter Nights* makes use of the *d20 System* for character advancement and statistics and as a basis for conflict resolution. Following suit, the *Aurora Toolset* allows authors to create encounters and creatures whose attributes are similar to the pen-and-pencil version of *Dungeons and Dragons.* Creating and editing the statistics of a creature or character in the Aurora Toolset is a familiar exercise for those knowledgeable in *d20 System* and *Dungeons and Dragons.* The similarity is illustrated in Figure 3.7 and Figure 3.8.

The *Aurora Toolset* employs the pattern BIND THE STORY in its use of a pen-and-paper style *D&D* character sheet as a metaphor for creature and character creation. Closely related is the use of the pattern IT GOES SOMETHING LIKE THIS in the *Aurora Toolset.* An 'encounter' is the terminology used for a group of creatures or

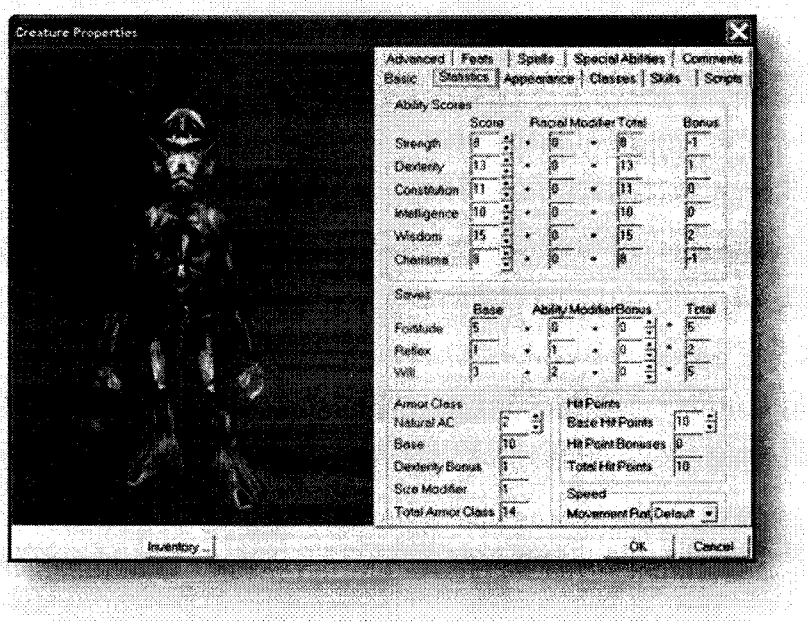

Figure 3.8: Creature Properties screen from the *Aurora Toolset*

characters that a player will meet and interact with at a particular point in the game. In the toolset, such a grouping of creatures is still called an encounter. The use of IT GOES SOMETHING LIKE THIS helps keep the pen-and-paper metaphor intact. While the *Aurora Toolset* does have elements of programmatic scripting, the dialogues themselves are written in the form of statements and responses as they would appear in the game, organized in a branching tree structure. This is a step in the direction of implementing TELL THE STORY IN YOUR OWN WORDS.

A possible disadvantage is the temptation for contrived use or overuse of this pattern. For example, 'Debug' is a term that does not seem to have a story-related equivalent, but is still used in *Inform 7*. To force a term to be domain-specific might cause confusion or a loss in meaning.

3.4.6 ALL STORYTELLING IS RETELLING

It is common for authors to re-use the components, characters and resources in their stories. Both ALL STORYTELLING IS RETELLING and TRADING STORIES address the organization of story components and their portability, respectively.

It is a difficult task to keep track of story resources, such as characters, components, text, titles, names and scripts. The problem is compounded by keeping track of multiple versions of the same resource, their locations and all the relevant details an author needs to know to effectively re-use. This places a mental burden upon the author or leads to the adoption of an informal secondary notation.

It is hard to keep track story resources, especially multiple versions and across multiple stories.

Therefore: Provide a common library of resources, preferably one that allows the addition of custom or edited content.

If you provide the environment for authors to create their own custom stories, then the environment should also provide a means of accessing common content and a place to store, access and organize custom content. Libraries allow authors to quickly access and add the common components to their interactive narratives. Libraries are common in programming, and are explicitly supported in environments such as *NetBeans IDE* and *Visual Studio.*

In the *Neverwinter Nights Aurora Toolset*, there is a library that game authors may use to access the blueprints for objects, creatures and map features that they may wish to use in their project. The blueprints make adding a dwarf to their environment quick and simple, as opposed to programming 100 lines of code for the "threatening little dwarf" as done in the *Inform 6* implementation of the classic interactive fiction, *Advent* [69].

A programmer may recognize a blueprint as a class and the use of a blueprint to create a dwarf in the environment as an instance of a class (an object). This domain-specific language is not required to convey the meaning of what is stored in libraries. The term library seems to be adequately domain-neutral. The blueprint is a simple enough concept to understand: two buildings may be built from the same blueprint. The model used in the *Aurora Toolset* is more similar to Alexander's pattern 205 STRUCTURE FOLLOWS SOCIAL SPACES [3], as it allows the user to edit existing blueprints to create objects and creatures that are unique to the story.

Disadvantages include the need to keep track of versions of resources and enforcing the distinction between a template (a class or blueprint) and an instance of the

template (an object). The edit properties screen, for example, in the *Aurora Toolset* for a template is almost exactly the same as the edit properties screen for an instance of the template. There is no visual indication, once in the edit properties screen, of whether the user is editing a template for the creature or an actual placement of a specific creature within the game world. Tracking versions of templates is left up to the user and is achieved by copying old templates and renaming them as new ones. Because characters and objects in a story change throughout the course of the story, over several modules for example, it would be most helpful to provide a means not only store these characters and objects but to also help keep track the changes or versions.

In addition, once an instance is edited, the only way to copy it is to right-click on the instance and "add to palette". This then creates a class from the edited instance from which more instances could then be created. This is rather awkward and is forcing the user to conform to a class-based programming model.

One lesser known programming paradigm, called prototype programming, may be a viable substitute for the concepts of classes and objects, at least as they are presented to the user in the *Aurora Toolset*. In the simplest prototype programming language model, an object consists of a state and behavior and is completely self-contained. Messages are sent to the object to request information about or change its state and behavior. To make a copy of the object, the object's state and behavior are copied, but there is no more relationship between the original and the copied object. This simple model would be expanded to include classification and to facilitate mass updates. This type of programming is suitable for visual programming, especially for less experienced users [78].

3.4.7 TRADING STORIES

While ALL STORYTELLING IS RETELLING addresses the organization of story components for re-use, the sub-pattern TRADING STORIES addresses how story components are shared. Authors of interactive narratives want to share their stories and the components of their stories with one another. Locating, importing from other computers and exporting story resources to other locations may be difficult.

It is difficult or costly to re-create the same resources in different stories or in different locations.

Therefore: Provide a means to import and export resources, thus supporting re-use, portability and the ability to share components of stories.

This is a logical extension of ALL STORYTELLING IS RETELLING, populating libraries with the content of others, and is illustrated by the large communities of interactive narrative players and writers that support resource libraries on the web.

The Neverwinter Nights Vault community is an example of this pattern, given that the *Aurora Toolset* also supports the importing and exporting of custom objects, encounters and creatures.

Adrift supports the import and export of modules and object scripts from one adventure to another. Wizards control the importing of objects to ensure that the scripting code matches that of the target adventure, and offers corrections to ensure that everything works as planned.

Designers should consider including a means to keep track of the components inside the exported resource. This could take the form of a secondary notation [45] system that is attached to an exported resource file, or the ability to browse and peek inside resources. A risk of sharing resources, especially over the web, is that with any file-based sharing system: you never really know what you are going to get. The risks are mitigated through the use of techniques similar to those used in anti-virus software, and by the nature of the self-policing communities that support the tool.

3.4.8 READ IT BACK TO ME

During the design process, authors commonly check their stories for errors, read through their creations to refresh their memories or to see if it reads properly. This is done continuously throughout the creation of the story.

Reading and re-reading a story throughout creation is common and often time-consuming.

Therefore: Provide a means to run the story within the STORYTELLER'S CORNER.

Similar to the code-test cycle performed by programmers of software, authors of

interactive narratives will desire to run their creations during the design process for a variety of reasons.

The most common reasons include testing (does my source code actually do what I want it to do?), debugging (are there any syntax or logic errors when I run it?), refreshing one's memory (where did I leave off last?), exploratory (what happens when I do this?) or simply wishing to view the latest part of their creation as a player in the story.

As seen in Figure 3.4, *Inform 7* has a 'Debug' menu as well as 'Go' buttons that allow the user to readily play through the narrative created from their natural language source code. The 'Go' catches and notifies of any syntax errors, as previously discussed. The 'Replay' and 'Play to Here' buttons give the authors the flexibility to continue play from different points in the storyline. These are not unlike the debugging, steps, watches and testing performed in programming environments like *NetBeans IDE* or *Visual Studio*.

The *Aurora Toolset* allows authors to run the module that they are working on, even if it is not complete. They may only, however, run a game module from the beginning. The consequence of supporting this design is that the system must have a strong sense of IN structures and story states. The *Aurora Toolset* does not. We examine this tool and discuss its limitations in Chapter 4.

3.5 Conclusion

With respect to serious game design, these patterns still apply. Whereas for entertainment games, this is all about building and supporting a community, for serious games this is about supporting creative writers and non-programmers, often subject matter experts in small organizations. STORYTELLER'S CORNER is the authoring tool that the user would use to leverage an existing game's engine and graphics libraries to create their own serious game content. EVERYONE HAS A STORY TO TELL begins to address the support need, for serious games, to open interactive narrative scripting to non-programmers. This pattern is broken down into three sub-patterns: TELL THE STORY IN YOUR OWN WORDS, BIND THE STORY, IT GOES SOMETHING LIKE

THIS that address the use of natural language, metaphor and story terms, respectively, to help the non-programmer use and understand the scripting environment. ALL STORYTELLING IS RETELLING, and READ IT BACK TO ME address two ways in which the serious game author needs to be supported, namely by providing easy access to story and game world objects and the ability to iterate through creations as they are being built. TRADING STORIES addresses the serious game author's need to share game resources and custom content.

Chapter 4

Assessment of a Commercial Tool

4.1 Introduction

In order to study and understand the support required for scripting interactive narratives, we conducted a study of a sophisticated tool. In this chapter, we present our research goal, examine and illustrate two frameworks in HCI and explain how they are used. We combine these two frameworks to conduct a study of a commercial end-user game development tool. And finally we present the results of the study and our conclusions.

In this section, we present our results of a Cognitive Dimensions of Notations [43, 44, 45] analysis, applied to *Neverwinter Nights Aurora Toolset*, a game scripting tool packaged with a successful video game. We borrowed from the approach used in Nielsen's heuristic evaluation [65, 75, 72, 73] in the application of the Cognitive Dimensions of Notations. The Cognitive Dimension of Notations framework itself, the advantages of this approach over heuristic evaluation and how we borrowed from Nielsen's work are discussed below.

4.2 Research Goal

The goal here is to examine a game scripting tool to increase our understanding of game scripting and the support required from such a tool. This is achieved by using a framework from human-computer interaction that grounds opinions and observations in recognized usability principles. Admittedly, the intent of the study is not to exhaustively find all usability issues with a particular tool, as one might perform during beta-testing or a re-design phase. Nielsen shows that this would have required at least 5 expert evaluators to find about 75% of the usability problems [72, 73]. Instead, we are interested in examining the scripting tool of a successful commercial video game

to capture salient ideas on the support required during the highly iterative process of creating interactive narratives. Vizri [104] conducted experiments that show there is no significant difference in either the number of usability problems found or the type of usability problems found when using low-fidelity vs. high fidelity prototypes (or the actual product) in studies that employ usability inspection techniques.

4.3 The Commercial Game and Tool

BioWare's *Neverwinter Nights* [10] is a computer role-playing game with a fantasy theme, set in the popular *Dungeons and Dragons* [46] campaign world of the *Forgotten Realms*. The game has a heavy interactive narrative component, with much of the game driven by dialogue between the player's character and the non-player characters in the game.

The game mechanic used in the game is based on the *d20 System*, used in the third edition rule set for *Dungeons and Dragons* [101]. A game mechanic consists of the rules used to determine outcomes of actions in a game, where the abilities and statistics of the characters and opponents, along with environmental factors, influence the probabilistic outcomes of those actions. *Dungeons and Dragons* is a game that is played on a table top, with pencils, paper, dice and often small figurines.

The commercial video game, *Neverwinter Nights*, was released with a toolset, called the *Neverwinter Nights Aurora Toolset*, which allows users to construct their own custom game worlds. The game allows the game author to act as the dungeon master, part storyteller, part referee, as other players play through their creations. Since its release in 2002, there have been six premium modules released by BioWare, two expansion packs and a number of editions. In 2006, *Neverwinter Nights 2* was released. Contributing to the success of the game was the ability to play online with up to 64 people and the ability to create persistent worlds on servers, creating essentially a small multi-player online role-playing experience.

In addition to its commercial success, the *Neverwinter Nights Aurora Toolset* itself has been utilized in serious games research. As mentioned in Chapter 2, Jenkins and a team at MIT used this toolset to build professional-looking serious games about American history with a low budget and a small team [31].

Following the lead Jenkin's team at MIT, Dormann et.al. also used the *Never-winter Nights Aurora Toolset* to create games that support informal learning about Antarctica [32]. Anecdotally, those involved in the project indicated some difficulties in using the toolset. We turned to Cognitive Dimensions of Notations to provide constructive means of examining the toolset and understanding any possible short-comings.

4.4 Heuristic Evaluation

Molich and Nielsen's heuristic evaluation is a "usability engineering method" [72] used to find usability issues in the design of a user interface. The method involves a small group of evaluators examining the given user interface independently and taking notes. Afterwards, a collaborative session is held and the individual notes are compared.

Evaluators normally make use of background information, list of tasks, use cases or scenarios, to work with the user interface. During the independent evaluations, the evaluators use a standard set of phrases representing "recognized usability principles" [73], called usability heuristics, as reference points for describing or explaining the issues found in the user interface.

Many usability heuristics for good design have been identified, but the general method does not advocate which ones to use. Nielsen, however, does offer a general set of widely used usability heuristics [73]:

- Visibility of system status
- Match between system and the real world
- User control and freedom
- Consistency and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use
- Aesthetic and minimalist design
- Help users recognize, diagnose, and recover from errors

- Help and documentation

Each usability heuristic is normally accompanied by a short description. For example, Nielsen presents the first usability heuristic from the above list as follows [73]:

> Visibility of system status: The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

Due to the succinct and operational nature of the heuristic phrases, the descriptions do not seem to be necessary beyond the first reading. This is one advantage of this approach. Another is that the reference points ground an evaluator's observations and opinions in the perspective of recognized usability principles. A third is that heuristic evaluation leverages inspection, placing it apart from prior observational techniques.

Our idea here is to take this type of heuristic evaluation approach and apply it as a method to studying game scripting tools. Unfortunately, the usability heuristics typically used in heuristic evaluation are much too general for the evaluation of an authoring tool for game scripting. Cognitive Dimensions of Notations, however, concentrates on notational systems. Notational systems are a special class of interface that involve a rich investment by the user, often have delayed feedback and are usually intended for long-term use. The *Neverwinter Nights Aurora Toolset* is much more than just a user interface; it is a notational system.

4.5 Cognitive Dimensions of Notations

4.5.1 Introduction

Cognitive Dimensions of Notations is a "framework for describing the usability of notational systems [...] and information artifacts" [12]. Examples of notational systems include spreadsheet programs like Excel, game scripting tools like the *Neverwinter Nights Aurora Toolset*, and ten-codes used by emergency personnel. Information artifacts are "self-contained notational systems". Examples of information artifacts include cellular telephones, portable media players (like the *iPod* and the *Zune*) and the controls for an air conditioner [12].

Cognitive Dimensions of Notations is similar to heuristic evaluation in that the goal is to identify usability issues with the given program. Cognitive Dimensions of Notations also provides a list of terms and phrases vocabulary, called cognitive dimensions or just dimensions, to be used by designers and evaluators. These dimensions, however, lead to discussions about the "implications of [...] design decisions" [12]. The implications are expressed as trade-offs, as the phrases are dimensional and a change in the degree of one dimension of a system will have effects upon other dimensions [45, 44, 43, 12].

While heuristic evaluation typically requires fully developed designs or mock-ups, Cognitive Dimensions of Notations may be employed at any stage of design and development. Problems can then be addressed as they are found (Green, 1998). In addition, the accompanying tools, aids or notations to the system being studied are also considered in a Cognitive Dimensions of Notations evaluation [43].

Furthermore, Green and Blackwell provide examples, illustrations, remedies, trade-offs and workarounds for most of the dimensions (Green and Blackwell, 1998). In other words, the Cognitive Dimensions of Notations method not only provides a checklist

of usability issues, but provides helpful information on how to successfully navigate, circumvent or mitigate the problems.

Green introduced Cognitive Dimensions of Notations in 1989 as a method to make comparisons between dissimilar interfaces and languages [45]. Green and Petre used the Cognitive Dimensions of Notations framework to compare three very different visual programming tools: *Basic*, *LabVIEW* and *Prograph* [44]. In an online tutorial, Green and Blackwell give detailed descriptions of the dimensions, trade-offs and known workarounds [43]. Blackwell presents a "state of the nation view" of ongoing Cognitive Dimensions activities, including a introduction, a summary of the dimensions in use and the rules a dimension must follow [12].

The essential rules are: a cognitive dimension is directional, like an axis. For example, something in relation to the dimension Visibility could be described as having a low, moderate or high Visibility. Also, cognitive dimensions should neither always good nor bad [45, 12]. Other important criteria for suitable dimensions include orthogonality (no two dimensions should describe the same phenomenon), granularity (dimensions must tackle problems at a similar scale) and naming convention (in as few words as possible, neutral, technical and accessible). Blackwell admits that creating new dimensions and names for them is difficult at best [12].

4.5.2 The Dimensions

The proposed cognitive dimensions are as follows [12]:

- VISCOSITY: Resistance to Change
- VISIBILITY: Ability to View Components Easily
- PREMATURE COMMITMENT: Constraints in the Order of Doing Things
- HIDDEN DEPENDENCIES: Important Links between Entities Are Not Visible
- ROLE-EXPRESSIVENESS: The Purpose of an Entity Is Readily Inferred

- ERROR-PRONENESS: The Notation Invites Mistakes and the System Gives Little Protection
- ABSTRACTION: Types and Availability of Abstraction Mechanisms
- SECONDARY NOTATION: Extra Information in Means Other Than Formal Syntax
- CLOSENESS OF MAPPING: Closeness of Representation to Domain
- CONSISTENCY: Similar Semantics Are Expressed in Similar Syntactic Forms
- DIFFUSENESS: Verbosity of Language
- HARD MENTAL OPERATIONS: High Demand on Cognitive Resources
- PROVISIONALITY: Degree of Commitment to Actions or Marks
- PROGRESSIVE EVALUATION: Work-to-Date Can Be Checked at Any Time

4.5.3 The Activities

The dimensions are not used in a vacuum; the designer or evaluator makes notes on the dimensional characteristics of a notational system or information artifact while considering the activity of the user. Blackwell and Green define four types of activities: incrementation, transcription, modification and exploratory design [43].

Activity: **Incrementation**

Definition: Adding further information without altering the structure in any way

Example 1: Adding a new card to a card-file

Example 2: Adding a formula to a spreadsheet

Activity: **Transcription**

Definition: Copying content from one structure to another structure

Example 1: Copying book details to an index card

Example 2: Converting a formula into spreadsheet terms

Activity: **Modification**

Definition: Changing an existing structure, possibly without adding new content

Example 1: Changing the index terms in a library catalogue

Example 2: Changing the layout of a spreadsheet

Example 3: Modifying a spreadsheet to compute a different problem

Activity: **Exploratory Design**

Definition: Combining incrementation and modification with the additional characteristic that the desired end state is not known in advance

Example 1: Typographic design

Example 2: Sketching

4.5.4 Activity Profiles and Workarounds

In Green and Blackwell's work, the cognitive relevance of the above activities is discussed with respect to some of the dimensions and they include activity profiles for each [43]. To illustrate, we consider the two dimensions, HIDDEN DEPENDENCIES and PREMATURE COMMITMENT, and their activity profiles.

The HIDDEN DEPENDENCIES dimension is defined as "Important Links between Entities Are Not Visible" [12] and as "a relationship between two components such that one of them is dependent on the other, but the dependency is not fully visible" [43]. An example of a system with a high degree of HIDDEN DEPENDENCIES is a spreadsheet program. In the activity profile for HIDDEN DEPENDENCIES, Green and Blackwell conclude that HIDDEN DEPENDENCIES have little effect upon the activities of incrementation and transcription, except when mistakes are copied. Modifying structure is often harmful, however, in systems with a high degree of HIDDEN DEPENDENCIES. Continuing with the spreadsheet example, changing a formula may have unforeseen ramifications elsewhere in the spreadsheet [43]. The problem stems from the design of spreadsheets, with formulas and data flow being mostly hidden from the user. This particular problem is examined at length by Igarashi [50] and

Panko [82].

Also included in the activity profile are common workarounds. Possible workarounds to systems with high degree of HIDDEN DEPENDENCIES include highlighting different information, providing visual cues when changes are made and providing additional tools for support [43]. Such workarounds are not always necessary, however. Design costs, trade-offs and the nature of the activity and the user may justify certain degrees of HIDDEN DEPENDENCIES. For example, simple unidirectional hyperlinks of HTML are still used. This means web designers and administrators rely upon additional tools, such as services provided by search engines, to tally the number of back-links to their sites despite the existence of the XML linking language, *XLink*, which makes multidirectional links between web resources possible [30]. In other words, HTML remains widely used and understood.

PREMATURE COMMITMENT is defined as "Constraints in the Order of Doing Things" [12] and "constraints on the order of doing things force the user to make decisions before the proper information is available" [43]. In the activity profile for PREMATURE COMMITMENT, Green and Blackwell state that a high degree of PREMATURE COMMITMENT is usually harmful with respect to all four activities. PREMATURE COMMITMENT also includes the cases where users are forced to look ahead in ways that are cognitively expensive [43]. When a designer has a differing idea from the user on the natural order of actions, this often constitutes a problem which may be explained by a high degree of PREMATURE COMMITMENT [43].

A simple example of a system with high degree of PREMATURE COMMITMENT is a cafeteria where the customer is presented with a choice of utensils before entering the serving area and thus knowing what is on the menu. Known workarounds to a high degree of PREMATURE COMMITMENT are decoupling, ameliorating and de-constraining [43]. De-coupling suggests using an intermediate stage. An intermediate

stage in our simple example might be to post a menu outside the entrance to serving area, near the utensil rack. But what if the customer changes their mind? Amelioration involves making bad guesses less costly to the user, which is essentially reducing the VISCOSITY of the system (see the dimension, VISCOSITY, above). Moving the utensil rack into the serving area may ameliorate the high degree of PREMATURE COMMITMENT. Unfortunately, cafeterias have lines and this may create poor traffic flow depending on the layout. De-constraining (or removing the constraints on the order of the actions) is another possibility. The utensil rack could be moved beyond the serving area altogether, close to the seating area, allowing people to get utensils at any time. Of course, the utensil rack should be visible (see the dimension, VISIBILITY) to the customers from the serving area.

The activity profile for PREMATURE COMMITMENT states that a high degree is harmful under all activities, which is a violation to the guidelines for a 'good' cognitive dimension [43]. Is it ever a good thing to be forced to make a decision before having the information necessary to make the right decision? Possible weaknesses of the framework, however, are the terms used for the dimensions themselves. Probably due to the requirements upon their names, the cognitive dimensions seem less accessible than the operational terms of heuristic evaluation. To overcome this, Green and Petre provide helpful questions for each cognitive dimension. PREMATURE COMMITMENT and HIDDEN DEPENDENCIES are presented in their work as follows [44]:

> HIDDEN DEPENDENCIES: Is every dependency overtly indicated in both directions? Is the indication perceptual or only symbolic?
>
> PREMATURE COMMITMENT: Do programmers have to make decisions before they have the information they need?

These thumbnail descriptions provide evaluators with pre-made questions to ask

themselves when investigating a notational system or information artifact. Also in their work, Green and Petre introduce the idea of 'straw tests' or measures of time that it takes to complete an action with respect to particular dimensions. They use this technique on a small set of dimensions to quickly and inexpensively compare three visual programming languages [44].

4.6 Study Setup

4.6.1 Select Task List

A set of online tutorials on module construction was selected as scenarios of typical use. The result of following the tutorials is a playable module, including three areas to explore: a town, an inn and a mine. The tutorial set was divided as follows [64]:

- Tutorial 1: introduces the New Module and New Area wizards and explains how to paint terrain.
- Tutorial 2: expands on the terrain painting skills learned in the previous tutorial and introduces new area manipulation commands. Painting and editing game objects is also covered.
- Tutorial 3: in addition to introducing doors, this tutorial completes the terrain painting lessons by explaining how to link areas together.
- Tutorial 4: describes the difference between the standard and custom palettes and demonstrates the different ways to create custom blueprints.
- Tutorial 5: explains Factions and introduces the Faction Editor in the Toolset. In addition, two new factions and a new creature blueprint will be created.
- Tutorial 6: introduces the Journal and Conversation Editors.
- Tutorial 7: discusses item creation and merchants.
- Tutorial 8: introduces ways to manipulate area lighting and sound settings.
- Tutorial 9: introduces the NWScript language and the Toolset's Script Editor.

The specifics of each tutorial will be described as required in the analysis section.

4.6.2 Select the Dimensions

A subset of all dimensions was selected in our Cognitive Dimensions of Notations analysis. A smaller set of dimensions is not only permitted but preferred, as it allows designers and evaluators to spend time on areas of interest. The result is a streamlined usability profile for the notational system [12]. The reasons behind each selection are presented here.

As previously mentioned, PREMATURE COMMITMENT is considered harmful with respect to all four activities and much of the work to be performed in the tutorial is akin to computer programming and the tool is in many respects like a programming environment. Green and Petre note with respect to programming "the order of working should be left up to the programmer, not prescribed by the environment" [44]. Programmers, while programming, frequently change the level of abstraction and textual order as they write code. It was surmised that this would be no different with game building. Modification is a common activity. It therefore is prudent to include a dimension that addresses potentially harmful effects with modification.

HIDDEN DEPENDENCIES are worth considering because of the many links between game resources, story elements and dialogue as experienced during game play.

VISCOSITY is an important dimension to consider, due to the nature of the tool as an editor. And "How much effort is required to perform a single change?" [44] is a valid question.

Despite their orthogonality, some dimensions seem closely related. VISIBILITY is closely related to PREMATURE COMMITMENT and HIDDEN DEPENDENCIES, and was thus selected as well.

Because of the iterative nature of game building and because game modules can be quite expansive, it is of interest to consider the dimension PROGRESSIVE EVALUATION and to answer the question: can "work-to-date can be checked at any time"

[44] by executing partially complete modules [43]? CONSISTENCY is often desirable in user interfaces. This dimension was not initially selected, but was later added as issues were found.

A designer of the *Aurora Toolset* writes [106]: The idea behind the creation of the Toolset is to allow one of the most important aspects of pen and paper *Dungeons and Dragons* to come to life on your computer — module building.

CLOSENESS OF MAPPING was selected to determine what "programming tricks" [44] need to be learned by world-builders to create adventures, creatures and encounters. Creating a *Dungeons and Dragons* adventure normally requires no programming skills.

Dungeon, for instance, is a magazine that publishes pre-made gaming scenarios, called modules or adventures, for use by game masters of *Dungeons and Dragons* [33]. A design guide located in the submission guidelines document for the magazine indicates the important parts of a *Dungeons and Dragons* adventure [34]. Some of the essential elements of an adventure include an introduction, a background for the adventure, a plot synopsis, a plot hook, descriptions of the creatures the players will encounter, descriptions of the areas they will explore, how to award experience points and a conclusion that discusses possible consequences for adventure's outcome. Optional parts include custom treasure, read-aloud text, traps, tactics, how to adapt the adventure to specific campaign settings and future developments to the campaign world.

The adventure writer must have knowledge of the game mechanics to fill out templates called 'stat blocks' for monsters, non-players characters, cities, objects and traps. These templates are the closest thing to computer programming that an adventure writer must do. It could be then argued that any additional programming requirement constitutes a usability problem.

In all fairness, however, in table-top role-playing games, it is the job of the game master (also known as the dungeon master or referee) to breathe life into these 'stat blocks', to describe what the player character senses, to control the actions of the non-player characters and monsters and often to supply ad-hoc, improvisational dialogue. The 'improv'-like nature of the game does not translate well into a video game [26, 41]. For now, we will consider "programming tricks" to constitute a usability problem if the system offers poor, little or no support.

4.7 Equipment

The following items were used in this study:

- A laptop computer running *Windows XP Professional* with *Neverwinter Nights Diamond Edition* (which includes the *Neverwinter Nights Aurora Toolset*) installed.
- A list of the selected dimensions, including their thumbnail descriptions.
- A list of other dimensions, including their thumbnail descriptions. These are kept on hand in the case a problem is found that was better described by an unselected dimension.
- Paper, pen and a word processor for note-taking and to store screen captures.

4.8 Procedure

The evaluation began with reviewing the dimensional terms and browsing the tutorial. Borrowing from heuristic evaluation, we followed the task list and the nine parts of the tutorial were worked through at the our own pace, using *Neverwinter Nights Aurora Toolset.* While working through the tutorial, we took notes, describing in detail any inconsistencies, problems and difficulties encountered, as well as

Dimension	Dimension Level	Issue Description or Thumbnail	Contributing factor, possible design decision or feature responsible
SECONDARY NOTATION	High	Comments about an exported resource are only available at end of import process	Import / Export feature
VISIBILITY	Low	Cannot view resources inside exported file without importing	Import / Export feature

Table 4.1: A Sample of Observational Notes by Dimension

observations that provided insight into the implications of the design. Each observation was either described using the cognitive dimensions or attributed to a cognitive dimension.

Further analysis involved all results being summarized and tabulated by dimension, degree, and the most likely contributing factor, feature or possible design decision involved. This is process is very similar to, and is borrowed from, the inductive process of building an affinity diagram in Contextual Design [9].

For example, Table 4.1 shows two observational notes, coded by the same contributing factor.

Placing codified notes in a table allows individual notes to be grouped together quickly by sorting the contributing factor column. From the table, and the groupings of the notes, interpretations are made and possible solutions are expressed. Not only is it a tidy summary, the table can serve as a quick lookup chart for any redesign or future design considerations. On successive iterations, one may also add 'priority' and 'severity' columns to the table to help make decisions on which items to address. Using an inductive approach to build such a table is our small, but novel contribution.

4.9 The *Neverwinter Nights* Model

Like many games, *Neverwinter Nights* has a software component called a game engine that renders graphics and handles game elements such as interactions, sounds and art. The engine is called the *Aurora Engine.* A game engine communicates with game scripts (that define interactions between player actions and game objects) via game events. The main advantage of a 'game engine' architecture is reuse. Once a game engine is in place, development efforts can concentrate on creating content (or game scripts).

Using the *Neverwinter Nights Aurora Toolset,* the user creates 'modules' that may be read and rendered by the Aurora engine. These modules contain the maps, creatures and dialogue that are in a particular adventure. The module file itself contains the scripts that define the game objects and their interactions. Each module is a .mod file. A .mod file encapsulates other files that in turn contain scripts for factions, journals, encounters, creatures, conversations, stores, items, situated objects (door and place-able object), sound objects, common game structures, palettes, areas, triggers, waypoints and multiple language support. Each type of script has its own format.

The *Neverwinter Nights Aurora Toolset* assists the user in creating scripts that populate the encapsulated parts of the module file. In some instances, the assistance is graphical (such as with painting areas) or through the use of template-like forms (like in creature creation). When the build command is used, the *Aurora Toolset* generates the scripts that populate the parts of the .mod file. In many parts of the *Aurora Toolset,* however, manual scripting is still required and is an essential skill in creating adventures [80]. An encapsulated resource file (.erf) is another file format used to pack multiple game-related scripts together. These files, .mod and .erf, are BioWare's way of packing multiple script files into a single file.

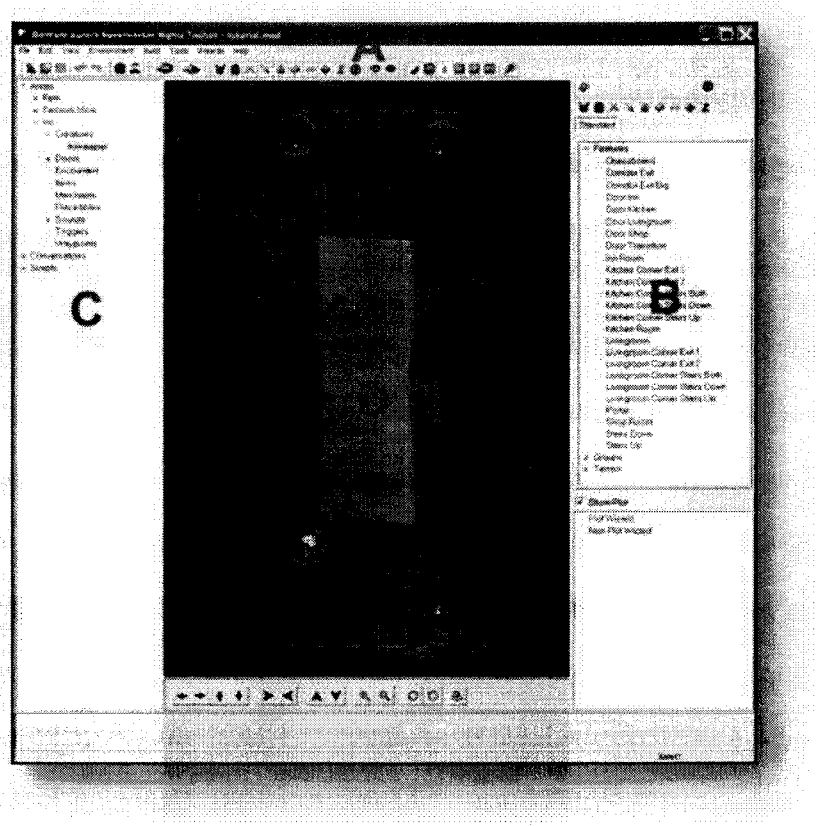

Figure 4.1: Screenshot of the *Neverwinter Nights Aurora Toolset*

The main difference between .mod and .erf files is that .mod files may be loaded and run by the game engine as a complete module. An .erf file contains game scripts, but is not executable. The user encounters .erf files when they export resources from a module, to be imported into another module.

4.10 *Neverwinter Nights Aurora Toolset*

The user interface of the *Aurora Toolset* is divided into four areas [106]:

A) Toolbar: commands, preferences and menus.

B) Palette: a hierarchy of content that may be added to a module. These include tiles, creatures, objects, doors and other objects. These are often referred to as blueprints in the tutorial.

C) Module Content: a hierarchy of objects that have been added to the module.

D) Area Display: the area that shows the currently selected area and where objects from the palette are added to the area.

4.11 Results

4.11.1 Premature Commitement

Constraints on the Order of Doing Things [12]

Do programmers have to make decisions before they have the information they need? [44]

Starting from scratch, the following constraint is seen: one must create a module and create an area before they can perform any actions in the toolset. The user is asked to create a module, provide a name for the module, create and area, provide a name for the area, select a tile set to be used in that area and decide on the size for that area. Resizing an area later on, discussed under Viscosity, is unfortunately a complicated task.

While there is a wizard that takes you through the necessary steps to create a new module, the wizard is not optional. The only way to get around using the wizard is to load an existing module, otherwise you will end up with an entirely grayed out screen.

But what if the user simply wants to create a resource (such as a new type of monster, a specific re-occurring character or a new magic item), but does not know where or in which module the resource will be used? The only way the system allows this is to create a module and map in which to house the resource. Later, when the user decides where to use the resource, they must export it to an .erf file and then import that file into the desired module. This problem is related to design decisions:

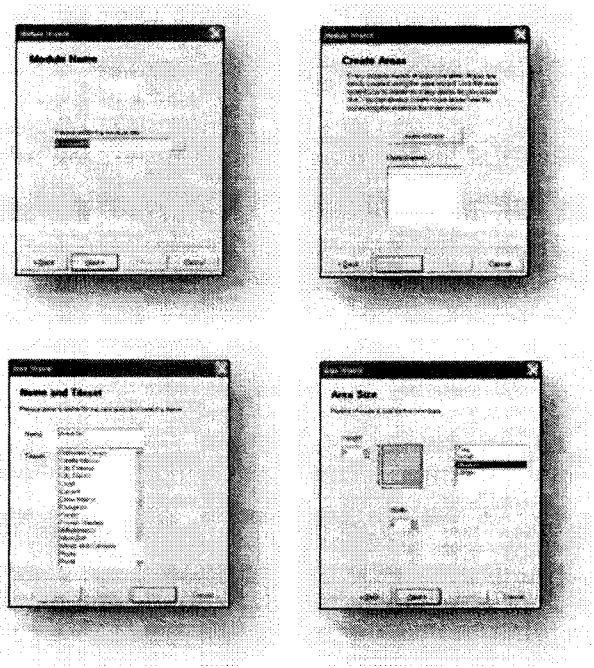

Figure 4.2: The first four steps in module creation

Figure 4.3: Screenshot of the *Aurora Toolset* showing the only two valid options at startup: create a module or open a module.

custom resources are module specific and a module and all of its constituent parts, including custom resources, are stored in a single .mod file. The high degree of PREMATURE COMMITMENT found here, stemming from design decisions, leads to a discussion about the VISCOSITY of the import and export tasks and the VISIBILITY, or lack thereof, of resources once exported to an .erf file and the resources in other modules.

4.11.2 VISIBILITY

Ability to View Components Easily [12]

Systems that bury information in encapsulation reduce visibility [12]

Is every part of the code simultaneously visible (assuming a large enough display), or is it at least possible to juxtapose any two parts side-by-side at will? If the code is dispersed, is it at least possible to know in what order to read it? [44]

Continuing from the discussion on PREMATURE COMMITMENT, the user is unable to view the resources inside an export file without importing it into a module first. The .erf file can contain multiple resources, but only their names are visible in the prompt given to the user when choosing which ones to import and potentially overwrite existing resources with the same name. A given resource in an export file cannot be viewed or edited until it is imported into a module. To complicate matters, entire area maps can be exported and imported between modules, but only the instances of objects (those 'painted' on the area map) are exported; the class-like 'blueprints' for a given object must explicitly be added to an export. This seems to be an issue with low CONSISTENCY. One can easily lose track of resources, object names, locations of blueprints, especially when hundreds of objects and blueprints are spread across even a handful of modules. Little support is offered to help manage this custom content.

The designers of the tool did try to provide some help, and provided a dialogue

box to fill in comments about a given collection of resources in an export file, a form of SECONDARY NOTATION. Unfortunately, the comments are not displayed until after the file is selected for import, and specific resources are chosen. Similarly, the user is unable to view resources located in modules other than the one currently open.

Other VISIBILITY-related issues include:

- Once open, there are few visual cues to differentiate between the edit screens of an instance and a blueprint; they appear almost identical.
- Local variables are not readily visible or accessible from the main user interface. The script wizard handles their creation and they are only accessible, after creation, in scripts.
- The script preview pane in the conversation editor only shows 6 lines of code. It cannot be expanded through conventional window manipulation.

4.11.3 VISCOSITY

Resistance to Change [12]

How much effort is required to perform a single change? [44]

From the *Aurora Toolset*, there is no way to add custom resources to the existing default list of standard resources. That is, custom resources that are created are stored in the .mod file of module in which it was created. When a new module is open, only the default creatures and items, for instance, are available in the Palette (see Figure 4.1).

In order to share custom resources between modules, one has to export the resource to a separate file (.erf), load the destination module and then import the resource from the file. Managing the location of custom resources and versions of those resources becomes a cumbersome task.

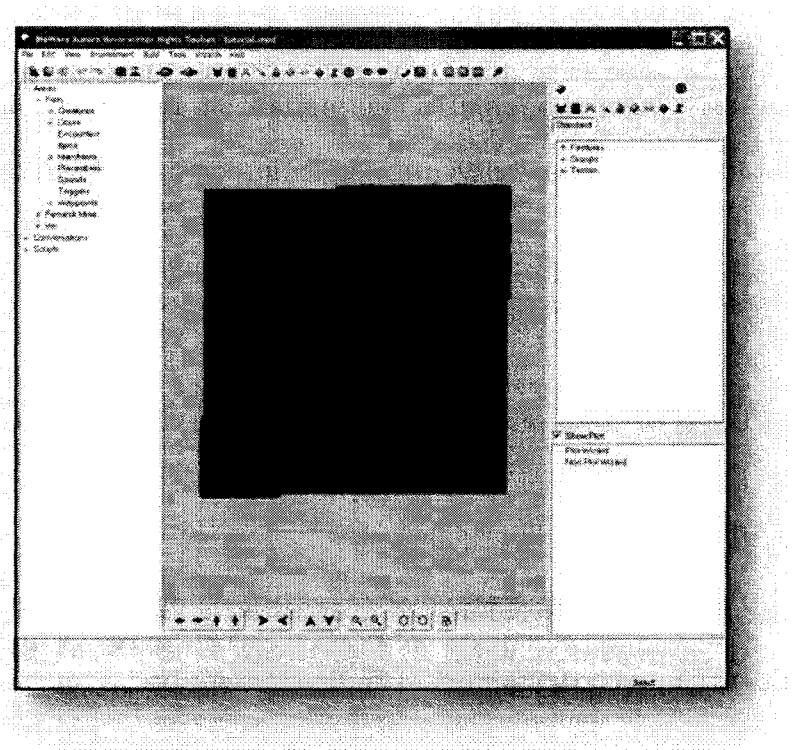

Figure 4.4: Screenshot of *Aurora Toolset* showing an area map and its grid of painted tiles.

Resizing an area map, a common modification activity, also requires effort. An area's size cannot be adjusted by dragging its borders with the mouse. The resize command is found in the edit menu and permits the user to modify the width and height of the map by adding or removing columns.

Through trial-and-error or by reading the warning in the tutorial [64], the user discovers that columns and rows are added (or removed) from the right-hand and topmost part of the map, respectively. Trimming or adding tiles to the left or bottom part of the map requires resetting the camera rotation to the default position, rotating the area map (which is subtly different from a camera rotation), resizing the map and rotating the area back to its original position.

The system does warn that deleting rows and columns will delete the items painted

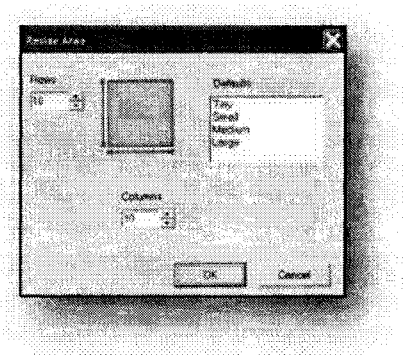

Figure 4.5: Screenshot of the pop-up dialogue box for resizing an area.

on those tiles. Instead of resizing, new areas can be created and connected by an area transition if an area proves to be too small or too cumbersome to resize. "An area transition is the means by which players are transported from one area to another" and "each area has a default load screen associated with it, based upon its tileset, which appears when players transition to that area" [64]. Usually, however, a load or transition in a game indicates a new set of objectives or a new level and may otherwise ruin the flow of the game.

4.11.4 CONSISTENCY

Similar Semantics Are Expressed in Similar Syntactic Forms [12]
When some of the language has been learnt, how much of the rest can be inferred? [44]

Problems related to a low level of CONSISTENCY are:

- Inconsistent availability of the Undo command. For example, you cannot undo an import, but you can undo the resize area command and undo objects painted on tiles.
- Inconsistent behaviour of the Eraser tool. It erases most terrain tiles, but for a few types of terrain tiles, like grass tiles, the eraser tool only toggles minor

variations in the look of the terrain tile.

4.11.5 Progressive Evaluation

Work-to-Date Can Be Checked at Any Time [12]

Can a partially-complete program be executed to obtain feedback on "How am I doing"? [44]

The build command in the *Aurora Toolset* generates the necessary scripts for the .mod file, flagging and notifying the user of any errors it encounters during the build process. Once built, users can test their creations by executing their modules and playing through the areas in the modules as a player would. This can also be done with incomplete modules, allowing for iterative design. There is a major drawback, however. A creator must play through their module from the beginning. Modules can be made up with many areas, contain hundreds of items, encounters and creatures. It is possible to use the same module for multiple quests and storylines. The inability to choose the point in the story or a game world state to start testing from makes testing a time-consuming task. The weak concept of state in the *Aurora Toolset* is a major factor contributing to this. The *Aurora Toolset* makes use of programming scripts that can, but do not always, define variables within them, to determine if game events have occurred. For example, in the tutorial, a local variable is created to store a value indicating whether or not a particular non-player character, named Falstadd, has been talked to before (*nFirstTimeTalked*). This variable was created in the Conversation Editor using a Script Wizard. This local variable was then referenced by a conversation script that was in turn attached to a node in the conversation tree.

A second script was written and attached to another node in the conversation tree. All this is done to make Falstadd have two different conversations with the player: he bestows a quest upon first speaking with him and later asks about the quest if spoken

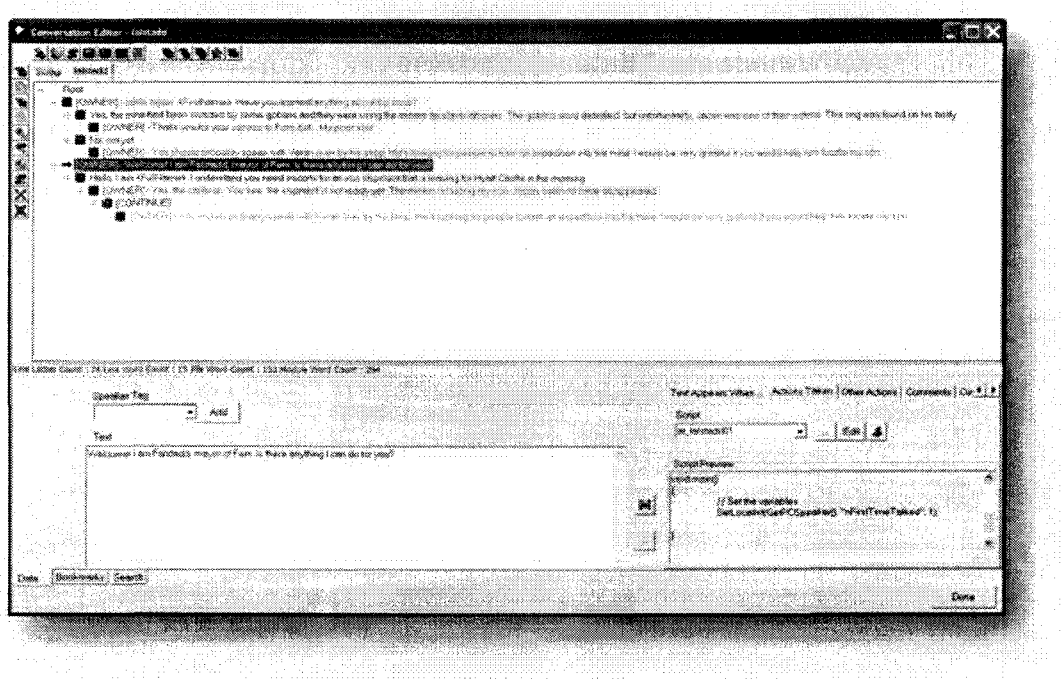

Figure 4.6: The conversation editor in the *Aurora Toolset*, showing the contents of a conversation resource called *falstadd* and the setting of a local variable called *nFirstTimeTalked*.

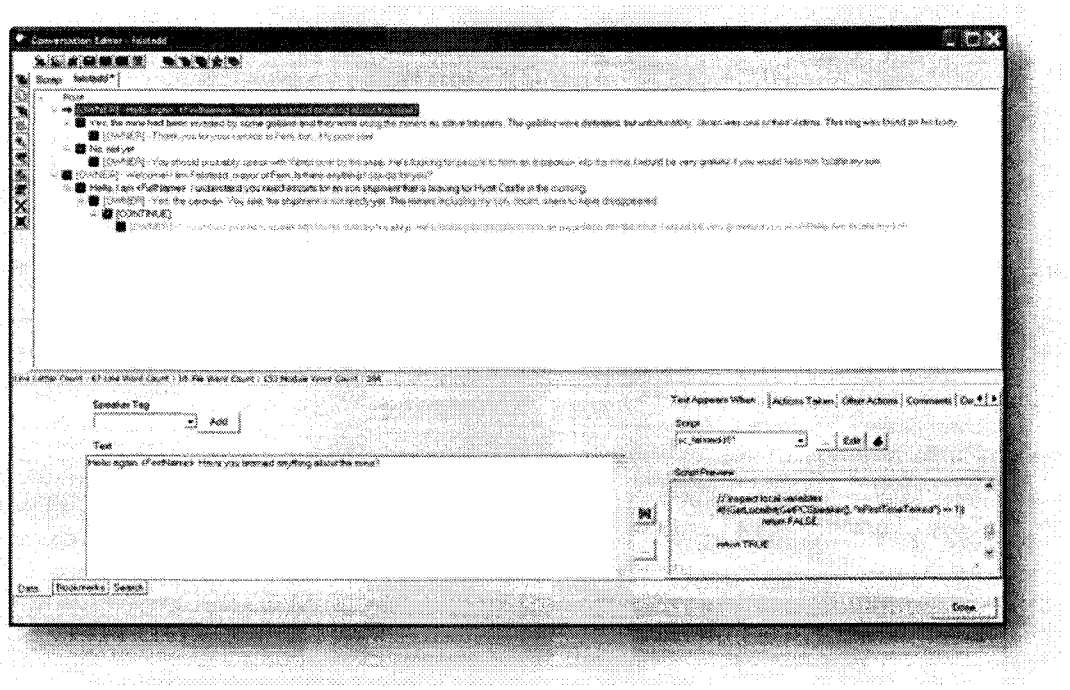

Figure 4.7: The conversation editor in the *Aurora Toolset*, showing the contents of a conversation resource called *falstadd* and the script used to check the local variable *nFirstTimeTalked*.

to again. Beyond the Script Wizard's help in creating scripts that defined and made reference to the local variable, the *Aurora Toolset* itself is not aware of the variable.

The variable, *nFirstTimeTalked*, is local to Falstadd's conversation only. The scripts that set and check the variable are only triggered when the conversation is initiated. The use of a local variable is suitable here, but what if it was desired to have this variable determine other outcomes in other parts of the adventure? In the tutorial, a third conversation is created, one that Falstadd has with the character once the player completes the quest. This one is handled differently, as the quest involves the recovery of an item (a ring). It must be present in the character's inventory for the third conversation to be available. When the item was created, it was given a unique identifier or tag. The conversation script is used to check the inventory of the character. No local variable is used.

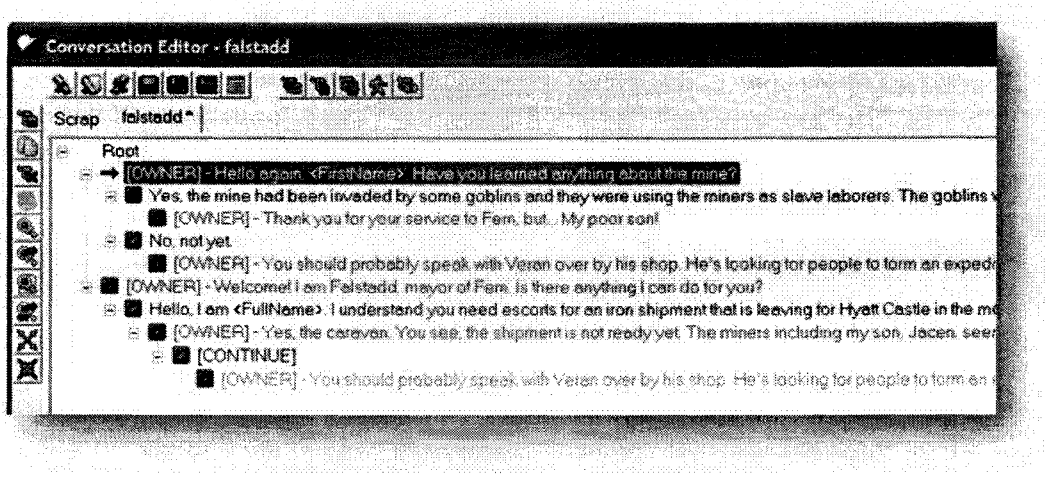

Figure 4.8: Close-up of the conversation tree for the non-player character labeled *falstadd*.

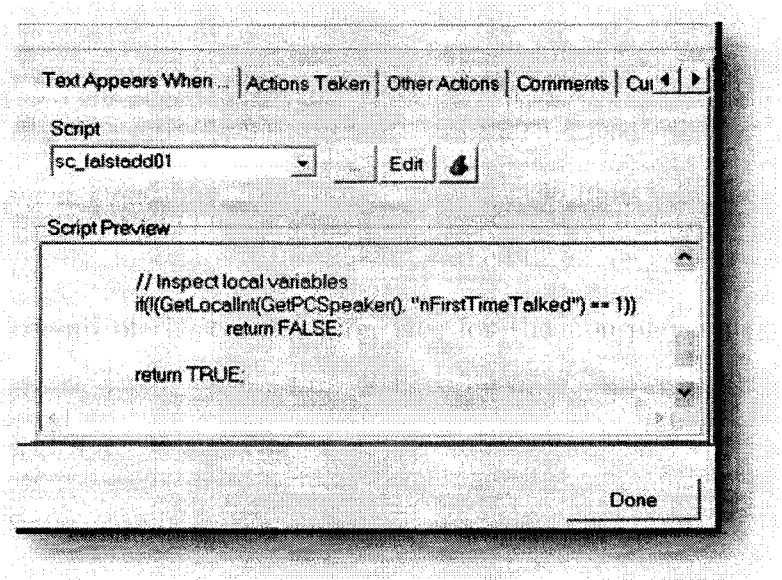

Figure 4.9: Close-up of the script in the conversation editor that checks to see if it is the first time the player has talked to the non-player character, labeled *falstadd*.

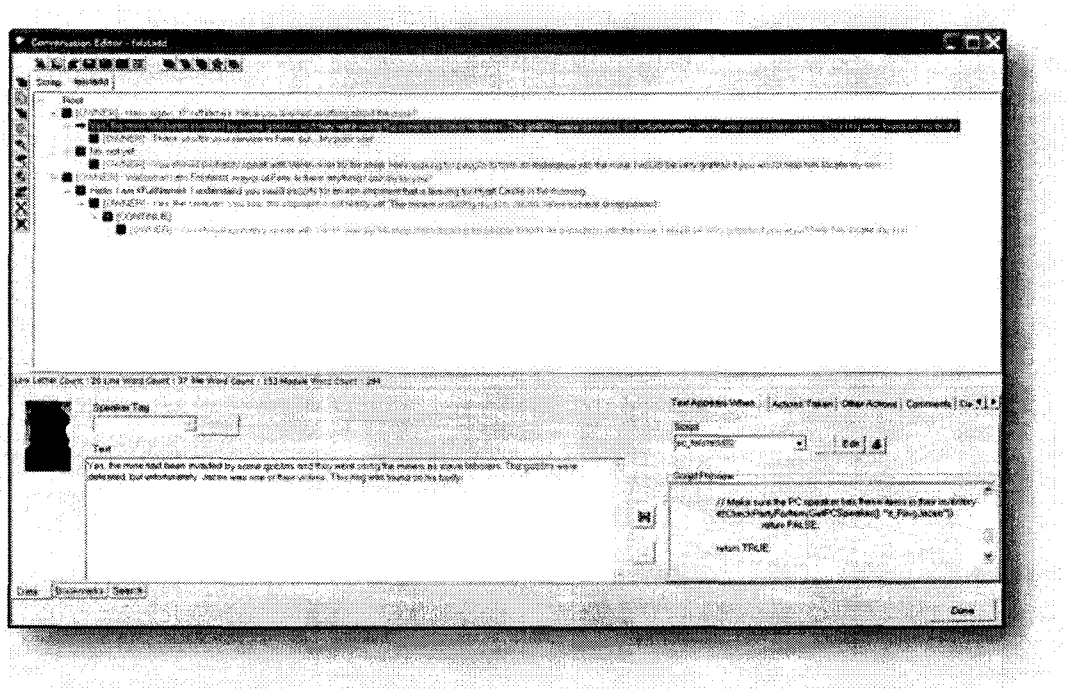

Figure 4.10: The conversation editor in the *Aurora Toolset*, showing the contents of a conversation resource called *falstadd* and the script used to check to see if the ring is in the player's inventory.

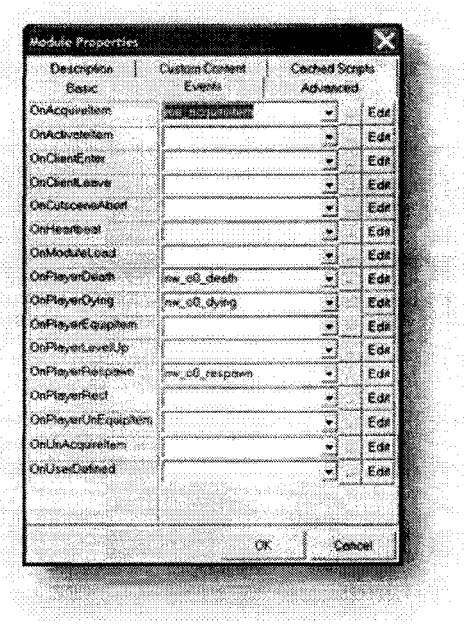

Figure 4.11: There are module specific events that may be defined using scripts. The highlighted script is run each time an item is picked up.

A module level script, one that checks to see if the ring has been acquired each and every time any item is picked up, updates the journal (another encapsulated part of the .mod file), instructing the player to return to Falstadd once the ring is found. This is the closest thing to a concept of global state for a module.

The player's journal contains text output based scripts triggered in the game, such as an event script for *OnAquireItem*. The text that appears is defined by the author in the journal editor. "[I]n general, the state of the plot or quest is measured [by the player] by its journal entries but is driven by the dialogue spoken by its participants" [64]. The hierarchical model of conversations and the fact that conversation nodes drive the state of the game implies that the interactive stories a user may create will be branching trees, foldback schemes or a combination of the two [26]. Therefore, the *Aurora Toolset* should provide support in creating, tracking and visualizing the states of the branching or foldback storyline. Unfortunately, this is not the case. Within

```
1  void main()
2  {
3      // variable declarations
4      object oPC;     // the PC that acquires the item
5      object oItem;   // the item that was acquired
6
7      // get the item that was acquired
8      oItem = GetModuleItemAcquired();
9
10     // if this item is a valid object
11     if (GetIsObjectValid(oItem) == TRUE)
12     {
13         // if the valid item's tag is "it_RingJacen"
14         if (GetTag(oItem) == "it_RingJacen")
15         {
16             // get the PC that acquired the item
17             oPC = GetItemPossessor(oItem);
18
19             // on the PC, set the journal with the tag "jt_Falstadd"
20             // to its second entry
21             AddJournalQuestEntry("jt_Falstadd", 2, oPC);
22
23         } // if
24
25     } // if
26
27 }
```

Figure 4.12: The script that checks for specific plot-important items whenever an item is picked up.

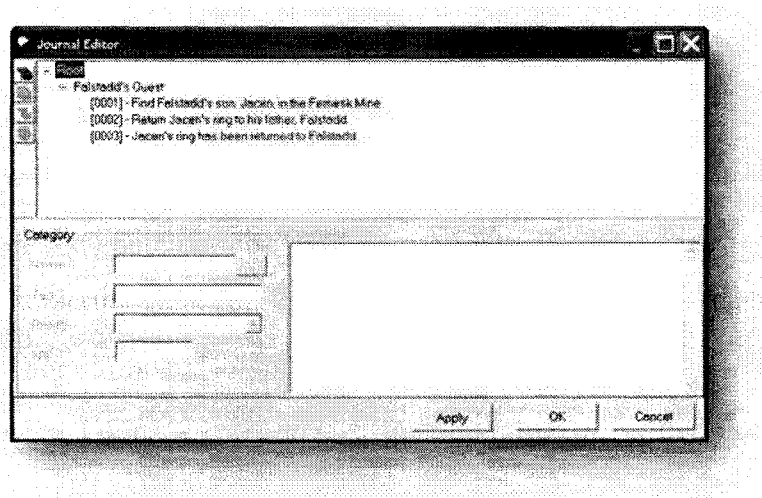

Figure 4.13: The journal editor, showing three lines of text that define the output to be written in the player's journal.

a module, there is a poor tracking of the state that matches the state of a game or story. While the states of a storyline are implied by the entries in the journal editor, these are only used by the *Aurora Toolset* as text output to the player journal, and not a means by which the system may utilize the state of a given storyline.

4.11.6 Hidden Dependencies

Important Links between Entities Are Not Visible [12]
Are there places where the user needs to resort to fingers or penciled annotation to keep track of what's happening? [44]

To compound the problem, in the journal editor, the author does not know which local variable or scripts trigger each journal update. Other issues involving high degrees of Hidden Dependencies are:

- Scripts listed in the Module Content Pane are linked to conversation branches, yet there is no indication from the Module Content Pane as to which conversations they belong. Editing the script directly does not indicate where it is used. One must open and edit the conversation resource and select each conversation branch to see the particular script that is attached. Builders must rely upon their own naming convention for scripts, notes or memory.
- When a builder edits an instance of a creature, there is no indication that the instance now no longer follows the blueprint for the creature. Conceivably, a builder could then edit a blueprint and update all instances accordingly, destroying previous edits made to instances.
- When dealing with custom resources that exist in multiple modules, there is no way of seeing which resource is newer or how they differ. Builders must keep track of differences between custom resources across modules themselves, or run

the risk of copying over newer ones during import. It is up to the user to make detailed notes during export in the dialogue box provided.

4.11.7 Closeness of Mapping

How closely related is the notation to the entities it is describing? [12]
What 'programming games' need to be learned? [44]

The *Neverwinter Nights Aurora Toolset* uses a collection of scripts to describe a creature's behaviour. While editable and configurable, the meaning of the existing scripts is poorly conveyed by the names of the scripts and their fields. An example script name for the field, *OnHeartbeat*, is *x2_def_heartbeat*. The user must learn from an outside source what the behaviour scripts do. The user must also learn how to read, edit and program scripts in order to change the behaviour of their creatures. As discussed under Progressive Evaluation, four scripts were required to handle a simple conversation, update the journal and check the inventory for an item. Such scripting requires a new skill set from creating table-top, pencil and paper role-playing modules or adventures.

4.12 Analysis

While the notes made at the time of the evaluation were described in relation to cognitive dimensions (see Appendix A), we present here a table ordered by the suspected contributing factors, possible design decisions or feature that led to the issue. As described in the Procedure, here are the tabled observations encountered during the study, the dimension that best describes the issue and the contributing factor or design decision that most likely lead to the issue. The table is sorted by contributing factor, illustrating how one design decision may impact the user experience in multiple ways. For example, the decision to have modules contain resources is the source

of a number of problems.

Dimension	**Dimension Level**	**Issue Description or Thumbnail**	**Contributing factor, possible design decision or feature responsible**
VISIBILITY	Low	Poor visual cues to differentiate between the edit screen of an instance and a blueprint	Code reuse
CLOSENESS OF MAPPING	Low	Must learn to program scripts to describe new behaviours or edit existing ones	Game creation requires programming experience
CLOSENESS OF MAPPING	Low	Must learn to program scripts to make non-simple conversations in the game	Game creation requires programming experience
HIDDEN DEPENDENCIES	High	Triggers for journal updates are not seen in the Journal Editor	Haphazard use of scripts and variable to track game state
HIDDEN DEPENDENCIES	High	Links exist between module scripts and conversation, but are buried in conversation trees	Haphazard use of scripts and variable to track game state
HIDDEN DEPENDENCIES	High	Links exist between module scripts and conversation, but are not visible in the Module Content Pane	Haphazard use of scripts and variable to track game state
PROGRESSIVE EVALUATION	Low	Can check work, but only from the beginning.	Haphazard use of scripts and variable to track game state
VISIBILITY	Low	Local variables are not visible or accessible from the main interface.	Haphazard use of scripts and variable to track game state

SECONDARY NOTATION	High	Comments about an exported resource are only available at end of import process	Import / Export feature
VISIBILITY	Low	Cannot view resources inside exported file without importing	Import / Export feature
PREMATURE COMMITMENT	High	Must create area before content	Modules contain resources
PREMATURE COMMITMENT	High	Must create a module before content	Modules contain resources
PREMATURE COMMITMENT	High	Must define area size before creating content	Modules contain resources
PREMATURE COMMITMENT	High	Must select tile set for an area without being able to preview tiles	Modules contain resources
CONSISTENCY	Low	Must explicitly add blueprints when exporting an area, but instance in the area are automatically added	Modules contain resources, Import / Export
HIDDEN DEPENDENCIES	High	System displays no concept date created or time last edited for resources	Modules contain resources, Import / Export
VISCOSITY	High	Many tasks involved in moving resources between modules	Modules contain resources, Import / Export
CONSISTENCY	Low	Undo command does not work for all tasks	Unconventional UI behaviour

CONSISTENCY	Low	Eraser tool does not behave the same for all tasks	Unconventional UI behaviour
VISCOSITY	High	Resizing the Map contains many overly complicated steps	Unconventional UI behaviour
VISIBILITY	Low	Script preview pane is too small. Cannot resize and must open up editor to view in larger window	Unconventional UI behaviour
CLOSENESS OF MAPPING	Low	Meaning of existing behaviour scripts is unknown	Use of external resource required

Table 4.2: Coded Observational Notes

4.13 Interpretation

From the above table, we see that issues fall into three general categories: poor behaviour/controls for user interfaces, an expected level of programming skill and, finally, features that would need to be changed to better support game scripting. One only needs to look at the behaviours of user interfaces from other popular commercial products as illustrations for improvement. For example, all panes in Macromedia's Dreamweaver are resizable. A 6-line preview pane for a program-like script is not useful, especially if you cannot resize the window. Or have a look at the canvas resizing controls in Adobe Illustrator. One would expect that the *Aurora Toolset* would support such programming-like endeavors, much like an integrated development environment, a programming tool that combines features such as compiler, editor and helpful tools to support programming. Instead, we find certain parts of the design the *Aurora Toolset* detrimental. Without a central repository for custom resources, world

builders are forced to resort to their own tracking methods and naming conventions to organize their custom content. This stems from resources residing in individual module files and export files.

While definitely not the intent of the designer, module creation in the *Aurora Toolset* remains a distant relative from that which takes place in pencil-and-paper, or table-top role playing games. On paper, module builders can easily borrow resources from rule books, source material books, online content and their own notes from past adventures. They can easily test pieces of their creations, like combat encounters, independent of a module itself. Unfortunately, in the *Aurora Toolset*, you cannot.

In addition, the overall design of the *Aurora Toolset* seems to display preconceived notions of what the tool should be used to create: a module that contains at least one area, as described previously under PREMATURE COMMITMENT. Recall that an area represents a place in a module, such as a building or a patch of wilderness. Areas are places where players explore, battle monsters and overcome obstacles in game-play. The builder's job is to populate these areas with (hopefully) interesting characters, buildings, items and encounters, everything with which the players will interact. The designer's importance placed upon areas implies that exploration should be a major focus of the games created with the tool. This may not be always desirable.

Yee categorizes the motivations of MMORPG (massive multi-player online role-playing games) players into the following three categories and ten sub-categories [110]:

- Advancement
  - Advancement
  - Mechanics
  - Competition
- Achievement
  - Socializing

  - Relationship
  - Teamwork
- Immersion
  - Discovery
  - Role-play
  - Customization
  - Escapism

Under Yee's categorization, exploration falls under the sub-category of discovery, a sub-category of immersion. While many players enjoy exploration, others may be motivated to play more for role-playing elements. Additionally, it is perfectly acceptable in role-playing games to have adventures that do not rely heavily on maps, instead concentrating on interactive narrative elements, role-playing scenarios or social interactions.

Take live-action role-playing (LARP) games, for example, where the players physically act out most the actions of their characters in an improvisational theatrical style. The setting for these games can be in private or public locations [90] and do not necessarily involve exploration and maps. A physical environment or location becomes the playing area, coined the *cage*: "An in-game mechanism that prevents the roles (not the players!) from leaving the game area during a play. Such mechanism can have a multitude of forms: physical, formal [or] social" [15]. Imaginary environments can be described or simulated by using dice, cards or improvisation. In Crawford's words, "Stories take place on stages, not maps" [26].

The most severe shortcoming of the *Aurora Toolset* revolves around the concept of game state. Game state is tracked in a combination of variables and programming scripts and the links between them are hidden and difficult to deduce. The system itself has little awareness of the state of a story or the game within a module, despite the use of branching structure of conversations that lends itself to developing

branching and foldback schemes.

The consequences are twofold. First, testing modules may be time-intensive. While the *Aurora Toolset* acts as a compiler and allows for testing a module under development, one must do so from the beginning of a module. Second, there is no visualization or support to create branching or foldback storylines, aside from individual branching or foldback conversations. The author is left with the cognitive burden of building and tracking the game state within a module without much support from the tool.

4.14 Related Work

The level of programming knowledge required to build modules using *Aurora Toolset* is a problem acknowledged and examined by researchers at the University of Alberta [80, 92, 27]. "Computer games typically have thousands of game objects need to be scripted" [80].

The main idea behind their work is to reduce programming to four types of scripts (encounter, behaviour, conversation and plot) and define templates for each. "The model is pattern template based, allowing designers to quickly build complex behaviors without doing explicit programming" [92].

Following the ideas behind generative design patterns [20, 38], their research group created a piece of software, called *ScriptEase*, as a companion tool to the *Neverwinter Nights Aurora Toolset* that generates programming scripts [92]. The program maps templates from their template library to the programming scripts required by the game. *ScriptEase* version 1.8.1, built in 2007, "supports encounter patterns" only [92]. Their behaviour, conversation and plot scripts are under development. It remains to be seen if these generative design patterns *ScriptEase* will replace most of the *Aurora Toolset*'s scripting support, perhaps such that users may only require using the *Aurora*

Toolset's area creation tools.

4.15 Future Work

With the release of *Neverwinter Nights 2*, a similar investigation using the Cognitive Dimensions framework could be applied to its toolset to see if similar problems are encountered, additional programming support was added and to identify possible workflow issues related to new innovations in game-play and the resulting complexity in game design.

4.16 Conclusion

We applied the Cognitive Dimensions of Notations framework like a heuristic evaluation to evaluate the toolset packaged with a successful commercial game. One evaluator used seven cognitive dimensions and identified twenty-two issues. We produced a table that grouped the issues according to the source of the issue. Such a table could be used in the re-design of the tool examined or as a reference for future designs.

Several key observations were gleaned in this study:

- The file system architecture had implications on the design and use of a tool. For example, encapsulating all resources needed by a module into a single file made managing custom resources difficult for the user. An editable library for both default and custom resources is a possible and desirable solution.
- Adventure/module writers for table-top role-playing games are familiar templates for monsters, cities and items. The use of template-like patterns that generate scripts for encounters, plot, conversations and behaviours appears to be a logical step in the evolution of game scripting support.

- The states of a game or storyline within a module were not understood by the system itself and were hidden from the user in a number of variables, events and scripts. Implications are:
  - Time-consuming development: the toolset was unable to allow testing of a module at any point of the plot or at a given state, forcing the user to test a module from the beginning. Given the iterative nature module building and scripting, this becomes very time consuming for large or very complicated modules.
  - Mismatch between models: There is no visualization, creation or editing support for branching/foldback storylines or game states within a module. The toolset falls short in providing this type of support, despite the model of what the user is expected to create.

While the *Aurora Toolset* itself is a tool specific to game scripting, there are parts of the tool that employ a language that is too general, with concepts from programming languages and environments. The observations, above, lead us to believe that replacing these parts with a domain-specific language [95, 30] may provide better support for scripting interactive narratives. An example of such support would be a visual language that allows the author to see the structure of the interactive narrative and how it relates to the structure of the plot and resources used in the game.

Chapter 5

Observing Professional Game Developers

5.1 Introduction

To futher understand the support needed to script interactive narratives for serious games, we conducted a study at a professional game development company that specializes in serious games and training software. In this chapter, we describe an HCI framework and detail how it was used to conduct the study. Finally, we present the results from observing employees at the company. We were permitted by the management of the company to conduct this research and we shared with them the results. For the purpose of reporting our results, and maintaining anonymity, we will refer to this company as GameCompany.

We used the Contextual Design framework [9] while observing and interacting with the employees at GameCompany. The end result included a checklist of observed issues, ideas for process improvement and design recommendations for internal tools. An advantage of utilizing this type of framework is that all issues, recommendations and ideas were based upon observations of employees as they performed their work.

The foci of our observations were the Game Programmer and the Creative Director, the specialists responsible for the code and narrative, respectively, in the games. In particular, we were interested in understanding how the Creative Director's work ended up in the game and what the Game Programmer's role was in narrative scripting.

Our results include a model of the work, represented by an affinity diagram, and recommendations for improvements in their process and tools. We also comment on the use of the contextual design techniques in this endeavor.

The research and findings presented in this chapter served as a pre-test for two other studies: one conducted by Marty Kauhanen and another by a fellow researcher in our group (see Chapter 6).

5.2 Research Goal

Through observations and interviews performed at a game development company, we sought to understand the issues present in game development, provide a list of recommendations to the company about their processes and ideas for internals tools, and improve our overall understanding of the game scripting. We sought to achieve this by forming a model of the work and identifying limitations of the tools and work process to seed design ideas for future tools for serious game design.

5.3 Contextual Design

5.3.1 Introduction

Contextual Design is a design process that uses customer data to drive the modeling and design process. First, one seeks to understand the customer, their work and their intent through inquiry and observation and then it is from this data that various models are created. The models are consolidated together to create a coherent and systemic view of the work, including its structure and the artifacts and roles involved. Key notes taken during the inquiry are coded and used in an inductive process to create an affinity diagram.

Not only does a clear understanding of the system of work result from the Contextual Design process, but design decisions that follow may be based upon the way customers do their work. In fact, in studying and understanding the work, one may also find opportunities to identify inefficiencies or even entirely new paradigms. Beyer and Holtzblatt define Contextual Design as a "process of discovering design implications for redesigning work practice, developing a corporate response and structuring a system in support of the redesign" [9]. Such an approach fits well with the established feature-driven design for software, as ideas can feed into the design or help prioritize feature lists. Design and feature decisions made in this manner are ground in user data, user intents and an understanding of the structure of the work practices.

The following Contextual Design phases were selected, as they are "good for a project that is interested primarily in identifying key issues of a product or in a user population" [48] and thus best fit the scope, resources and purpose of the practical study.

The phases are:

1. Identify subjects and the work to be observed
2. Contextual Interviews with Interpretation
3. Affinity Diagrams
4. Wall Walk and Visioning

Each of these phases is discussed below, along with the results of each phase.

5.4 Equipment

A digital audio recorder was utilized in this study to capture key parts of the interviews. Since the contextual interviews took place at the participants' place of work

and at their work stations, no additional setup or equipment was required.

5.5 Subjects and Workplace

Prior to the interviews themselves, site visits to GameCompany were conducted. General information was collected about the company and from interview candidates. The profiles of the company and the two employees selected as subjects are as follows:

GameCompany Profile: An Ottawa-area software development company with 15-20 employees, most of whom have university-level educations. The company creates interactive, serious games for professional skills training, certification and assessment. The content of the games includes government and industrial standards and emergency response plans. The games are *Flash*-based and some have elements of assessment included in them.

Creative Director Profile: The roles of this position include creative writer, member of the creative team, liaison with customers and marketing. Creative Director is responsible for producing creative content and narrative for the serious games, providing input for the games and providing direction to the graphic artists. She sometimes works from home and alternates between a desktop computer and laptop computer. Creative Director uses *Microsoft Word* on a regular basis.

Game Programmer Profile: His roles include programmer, technical advisor to creative team, integrator of game components and researcher and designer of new components. Game Programmer works primarily at his office space and uses *Flash Developer*, *oXygen* (an XML editor), *Open Office* and *Internet Explorer*.

Other members of the development team include: Graphic Designer and Sound Designer. The Graphic Designer's principle responsibility was to design and create the visual flash animations for the game. Working closely with the Graphic Designer, was the Sound Designer, who was responsible for the audio components of the games.

Their 'creatives' were incorporated by the Game Programmer into the games.

5.6 Results

5.6.1 Pre-Interview Sessions

In the months prior to the taped contextual interviews themselves, there were regular meetings and site visits performed by our research group to GameCompany to familiarize ourselves with their products, workspace and employees. It was during these meetings that we identified the subjects and the work to be observed. In addition to the site visits, there was correspondence via email and over the phone to schedule the interviews.

5.6.2 Contextual Interviews

We met with the each of the two participants to talk about the study, their roles and backgrounds and about the work that they perform. Parts of the contextual interviews were audio recorded, including two 2-hour sessions with each participant, while they performed work pertinent to serious game creation. Each session was on a separate day, to make sure that we witnessed work pertinent to the study and to minimize us disturbing their productivity.

The interviewer observed and spoke with Creative Director and Game Programmer while the subjects performed their work. The interviewer adopted apprentice-like and recorder roles during the interview. They were permitted to ask many questions about the work as the work occurred.

5.6.3 Interpretation Notes

From the notes taken during the interview and from the transcripts, key interpretation notes were made. These notes capture key issues about the data and were used to

build an affinity diagram. The interpretation note is tagged by subject and number. Here are a few examples of interpretation notes from the interviews with the Creative Director and some additional explanation for each:

Creative_Director- 32 Edits existing stories in Word (not in XML)

The Creative Director does most of her work in Word. She keeps track of all characters, their descriptions, actions and dialogue in document files. Also included in her document files are notes, ideas and other material that does not necessarily make it into the game itself or even into her final version of the narrative content. From these document files, she creates 'clean' versions of these files for the Game Programmer to incorporate (re: cut and paste) into the game file. If an edit needs to be made, she edits her document file and then passes it on to the Game Programmer.

Creative_Director - 33 Different game methods require different narrative structures

There are a few different types of games that GameCompany creates, some more complicated than others. She uses different headers for different types of games. This is still far from a template, and more so an organization practice for herself.

Creative_Director - 34 Subject tries to add story elements into the game so that it isn't just e-learning.

The creative director always tries to balance fun and interesting dialogue with the learning requirements.

Creative_Director - 35 Wants to add narrative directly into game by themselves

> The creative director realizes that it would be easier for everyone if she could edit the code directly. She does not feel comfortable being creative and dealing with structured tables, however, and does not want to learn how to read XML.

And here are some interpretation notes from the interviews with the Game Programmer:

> Game_Programmer-07 Keeps hard copies of documentation at desk
>
> The game designer keeps a number of specification documents scattered around his workspace. They contain notes from meetings. These specifications are also located on a shared drives, but not all the documents are useful and he wants the ones he needs on hand. He does not like the quantity of documentation and it is not always relevant or helpful to him.
>
> Game_Programmer-08 Creates low fidelity prototype on paper or whiteboard
>
> Initial prototypes and game ideas and layouts are drawn on the whiteboards located at each desk and in the design team's shared workspace.
>
> Game_Programmer-09 Create high fidelity prototype in Flash
>
> Flash prototypes are made for mini games, or games within a game. These prototypes usually end up being edited and included as part of the game.

Also present amongst the interpretation notes are special notes that captured design ideas (denoted by DI in the interpretation notes) that were discussed during the interviews. For example, when interviewing the Creative Director, the subject

explained that they would like a way to organize and keep track of the character names, titles and locations within game levels across multiple games. The following two design ideas were discussed in the interview:

> DI Creative_Director-22 Wants to sort characters names by ethnicity, popularity, and date
>
> DI Creative_Director-21 Database of character names and titles
>
> The Creative Director agreed that a centralized database that contains character information (name, title, ethnicity, locations within the games, and personality) would be useful and helpful, as opposed to a multitude of document files.

Approximately 150 interpretation notes, including 14 initial design ideas, were taken and used in the creation of an affinity diagram (see Appendix B).

5.6.4 GameCompany's Game Architecture

An internet browser and internet connection is required to play a game. The browser downloads and runs the game client. The game client loads game files on the fly, handles the variables and finite state machine, acquires and loads the art, sound and Flash resources for the game, as defined in the game file. The game client also handles all other communication to the game server. For some games, this communication includes tracking and logging the user's performance in the game for assessment purposes. The game code itself is written in XML (extensible markup language) [105].

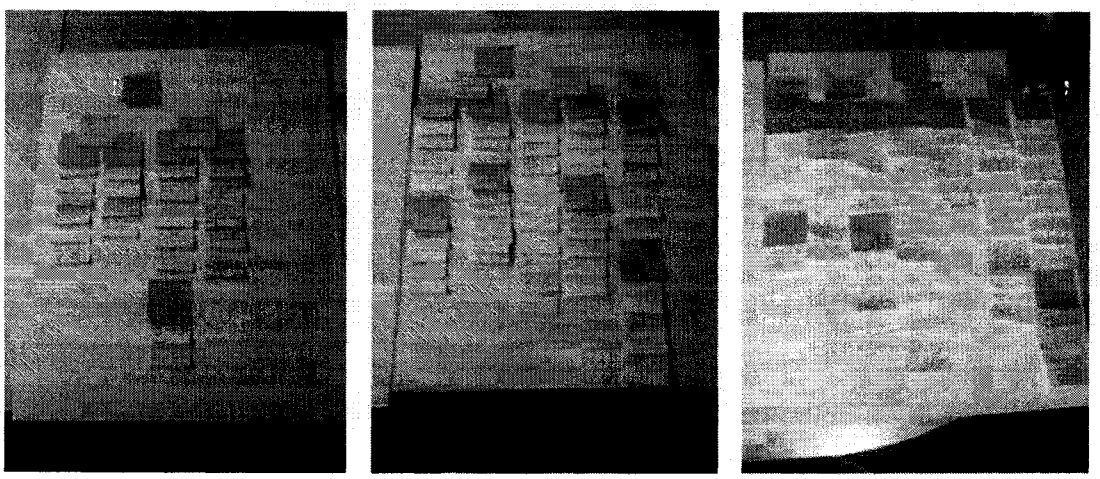

Figure 5.1: Three pictures of the affinity diagram, one for each inductively created top-level heading.

5.7 Analysis: the Affinity Diagram

The affinity diagram is an essential step in all the Contextual Design processes. Building an affinity diagram itself is an inductive process of tagging and grouping the interpretation notes (including design ideas) by likeness. The like groupings are then grouped together, inductively, into categories and the categories are grouped until one arrives at handful of abstract and categorical groupings. The result is a hierarchical grouping of the interpretation notes that represents the practices of the user population [48].

Note on Figure 5.1: Our red, orange, and pink notes correspond to the following colours used in Contextual Design: green, pink and blue, respectively. To avoid confusion, the results of the affinity diagram below were presented using the same colors as the sticky notes used in this study. The pink-level labels tell you what is underneath them, characterizing the work, the users' perspective and any issues that the design team needs to consider. The orange-level labels may reflect the larger steps of the process, communication strategies and how the tools are used. The red-level

labels the central story of the work and are categorical or abstract. It is important to note that the headings are created using an inductive process [48].

One may read the affinity diagram top-down, however. From our interpretation notes, we induced three abstract red labels: 'Strategic Creativity', 'Teamwork' and 'A Highly Technical Activity' that capture the essence of the work performed by the members at GameCompany.

From these three categories we know that serious game creation is an activity that is highly technical, is a team effort and requires a balancing act between creativity and learning requirements. The fact that the three green labels capture the essence of the work gave us confidence in the Contextual Design methodology. The affinity diagram indeed helps us with the goal of this study, namely, to understand the work. The orange and pink labels will help us articulate the intents of the developers and, along with the interpretation notes themselves, provide insights towards recommendations for tools and aid in their process.

The labels of the affinity diagram, inductively created from the interpretation notes, are presented below. Please note that 'I' and 'We' in these labels represent the perspective of the employees interviewed at GameCompany.

- Red: A Highly Technical Activity
  - Orange: Programming Is Involved
    - Pink: I Don't Like Documentation
    - Pink: I Debug
  - Orange: The Work Needs To Be Visualized
    - Pink: Computers Are Not Always Used
    - Pink: The Game Is Layered With Respect To Components And Graphics
    - Pink: I Want A High Level View Of My Work
  - Orange: Existing Tools Have Some Of The Desired Features But Some Tasks Could Be Automated

    * Pink: Existing Tools Do Some Of The Things I Need To Do My Work
    * Pink: I Want To Automate Some Tasks
  - Orange: Tools Should Help Component Re-Use
    * Pink: I Want Tool To Be Aware Of Resources And Components
    * Pink: I Want to Re-Use Components
    * Pink: The Type Of Game Determines The Structure Of The Game
- Red: Teamwork
  - Orange: Development Is A Team Process
    * Pink: Communication is Informal
    * Pink: We Work On Games Simultaneously
    * Pink: We Work Collaboratively On All Aspects Of A Game
    * Pink: People Check My Work
  - Orange: We Share Our Files
    * Pink: We Organize Our Files
    * Pink: We Keep Track of Changes
    * Pink: I Keep Track of Characters
  - Orange: Team Members Have Different Roles
    * Pink: Some Work Is Done In Order
    * Pink: Work Is Divided
- Red: Strategic Creativity (Balance between Requirements, Learning and Fun)
  - Orange: Every Game Has Essential Requirements
    * Pink: I Need To Know The Requirements For the Game
    * Pink: I Need To Know the Subject Matter
  - Orange: Creativity Is Important
    * Pink: I Create Characters
    * Pink: I Sketch and Save My Ideas
    * Pink: I Need To Be Creative

The full affinity diagram, including the yellow interpretation notes, is included in Appendix B.

From the labels of the affinity diagram, we see the that developing a game is a team endeavor that relies upon informal communication, shared ideas, shared resources, coordination, different roles and the need to be organized. The team members work on their own aspects of a game simultaneously and collaborate often.

We also learned that a game's interactive narrative is structured and some of the tasks are not supported in third-party tools, because they don't recognize that structure. In addition, both the the Creative Director and the Game Programmer used whiteboards and hand-drawn diagrams to visualize the IN structure.

A gap exists between *Word* and *oXygen*, as there is no easy way to insert narrative into the game. Drawings on whiteboards and paper are used to visualize the states of the gameplay and the story structure. The final *Word* file also contains the structure of the game, using headings and titles as a means of organization.

The Creative Director values creativity and WYSIWYG interface of *Word* for writing characters, dialogue and narratives. She stores her creations in various files and keeps track of her own versions. When it comes time to insert narrative structure into the game, the Game Programmer manually translates the chapters into code. Even more cumbersome and time consuming is when dialogue is added. The Game Programmer must cut-and-paste the Creative Director's text into the XML files.

5.8 Interpretation: Wall Walk and Visioning

A wall walk, also called a data walk, involves going through all the data presented in the affinity diagram to come up with design ideas. The new design ideas are written down and posted as new labels on the affinity diagram. Any holes in the diagram are plugged with more data, written on labels. Wall walking may be the last step in the Contextual Design process if a given organization has its own design process [48]. Right after the walking exercise, a list of key issues and hot ideas should be made.

A wall walk and visioning session, involving primarily the two observers, resulted in a list of key considerations and a number of "hot ideas" [48] for design of internal development tools.

5.8.1 Key concerns

- Communication between narrative in document files and the XML code is disjointed.
- Adding dialogue to the XML code is an iterative and ongoing process.
- Documentation is in multiple locations and in many formats. It is not always written in a format useful to the Creative Director and the Game Programmer.
- Debugging is time-consuming and a common task. An organized approach and support is needed.
- Use of 3rd party tools is highly prevalent. Current tools do some of the things that are necessary to get the job done.
- *oXygen* provides no support in matching tags in the game code to the game resources, such as sound or art files.
- Balancing creativity and fun with learning requirements is difficult.
- The creative director needs support for managing characters and character information.
- Games of the same type have a common structure. There is a need to visualize this structure.

5.8.2 Ideas for internal software tools

- A software tool to support the task of transforming documents that contain characters, dialogue and their actions written by the Creative Director into the XML game code.
- Changes made to the dialogue through the tool should be automatically updated in the code, without cut-and-paste procedures.
- A major consideration is that the Creative Director still would like a WYSIWYG type interface for writing; she does not want to see any code.
- Multiple views in the authoring tool show the desired parts of the code, using filters, depending on the user's preference. For example, the Creative Director would see her notes, comments, and the dialogue, whereas a programmer's view would include code, reference to resources and the like.
- The code is the documentation, as it contains requirements, comments, narrative, code and the game logic.

The tool should also:

- automatically create the required format for the game files, depending on game type (for example, assessment vs. non-assessment games)
- provide drop-down menus for game resources and links them to XML tags
- provide an editable graphical interface for representing game logic and story elements

5.9 Conclusion

The above interpretation notes, affinity diagram and interpretations were presented to the company. We suggested prioritizing the key issues and have a larger team to perform a wall walk and visioning session, one that includes members of GameCompany. While we were not present to observe the discussions on how the company used our results, future site visits to the company would reveal that employees were taking actions to address some of the issues identified in this study.

Chapter 6

One Year Later at GameCompany

6.1 Introduction

The value of the Contextual Design process, in identifying key issues, is exemplified through another formal study at the same game company in 2008, a year after the contextual inquiry was performed. When we returned, we discovered:

- a bug-tracking system in use
- two *Word/Excel* and XML conversion tools, developed internally, for internal use
- early design plans for an advanced script generation tool
- an usability specialist on staff

The fact that the company addressed many of the concerns and issues raised in this study illustrates Contextual Design as a valid research tool for examining, understanding and improving software development and, specifically, game scripting and supporting tools.

The item of particular interest was the development of conversion tools to address the gap between *Word* and *oXygen*, between the Creative Director's work and that of the Game Programmer. Remember the long term goal of GameCompany: to allow subject matter experts to create games and update game content via authoring tools. In other words, automating or supporting narrative scripting is an essential step to realizing this goal.

In this section, we present the results of a cognitive walkthrough and usability testing on two internal tools developed by and for employees at GameCompany.

6.2 Background

GameCompany employs an incremental and iterative development cycle. During the planning and requirement phases, requirements are gathered from the customer, SMEs are consulted. In the analysis and design phase, user requirements are examined; designs are drafted and reviewed with the customer and SMEs.

The implementation phase involves several people working simultaneously and incremental integration. The Graphic Designer, Sound Designer, Creative Writer and Game Programmer all work simultaneously on Flash animations, sounds, non-player dialogue and game code, respectively. The Creative Writer is both a different individual and a different position from the Creative Director, but is also involved in writing the narrative content for the games.

The Game Programmer would include sounds, Flash and dialogue, as they were ready, into the game that would appear in successive builds of their game. Some resources, like scrollbars and dialogue boxes were re-used from previous games. Other resources, like dialogue scripts, are game specific.

Due to the highly iterative nature of describing non-player characters, actions player characters and writing dialogue for the game, the process of updating the code to reflect those changes is a common and regular task. Play-testing was performed regularly and minor edits were common.

In addition, the games created are serious games, designed for the use as training aides for government and industry standards. As such, new requirements for dialogue and descriptions are often found during the testing and evaluation phases of the iterative process.

One problem that was captured during the contextual interviews (see Chapter 5) was a gap in communication between the creative writer and the game designer, in particular between the *Word* files that contained the creative writer's work and the XML code that was the game designer's domain. For the CreativeWriter, it involved keeping 'clean' versions of the game dialogue and character descriptions that did not contain notes, comments and ideas that commonly populated her working files (in *Word*). For the Game Programmer, it meant adopting a cut-and-paste strategy to transfer content of document files to the XML game files and managing versions of document files along with versions of the code.

In the year following the contextual inquiry, software engineers at GameCompany created two software tools for internal use in an attempt to address the gap between the creative writer and the game designer. This problem represents an obstacle towards achieving their long term goals.

6.3 Goals of GameCompany

A long-term goal of GameCompany is to provide end-user development tools so that customers could easily create the content for their own games. The gap between the narrative and code was one such problem that needs to be solved in order to present a usable content creation, or even a game-creation system, to non-programmers.

Management of GameCompany wants to enable people, such as customers and subject matter experts, who are not familiar with XML, to make game content in some form that is easily readable and converted into a game file. Their strategy was to leverage the common familiarity of *Microsoft Word* and *Excel* and build two software tools: one that converts document and spreadsheet files into XML and another that converts their game XML back into either a document or a spreadsheet.

The two conversion tools, in the alpha stage of development, were being used by the game development team to produce both XML based on creative content documents and document files based on the XML in the data section of their game code.

An initial cognitive walkthrough, a usability inspection method, performed by Marty Kauhanen, Minh Tran and Robert Biddle, indicated the need for a usability study [75, 74, 47]. The cognitive walkthrough revealed that the conversion software was neither intuitive nor easy to learn and understand. A particularly confusing part of one program involved creating rules that defined how the document file was to be converted into XML file.

The employees of GameCompany welcomed the usability study as feedback to help identify improvements for the next version of these tools and for future tools. At the time of the visit, the game development team was working on how to best incorporate these two tools into their design and development cycle and is considering plans to automate the conversion processes. In addition, they were in the initial planning stage to develop programming tools that allow editing of the control and consequence sections of the game script code, in addition to the data section where the converted dialogue is stored (see next section for description of a game file).

Unfortunately, a stringent non-disclosure agreement with GameCompany prohibits screen captures and pictures of these two tools. This is largely due to the prospect of commercial release and the sale of a version of these tools as generic XML-to-*Word*/*Excel* and vice-versa converters.

6.4 GameCompany's XML Code

The game code is written in XML that is read by a thin-client game client.The XML game script itself is divided into five sections: resource, variable, data, consequence and control sections. The resource section is where a programmer defines many of the assets that the game will use, including things like re-usable scrolling objects and menus, and links to the files that contain graphics and sound. The data section includes static global variables, like configurations for menus. It also includes a sub-section for game-specific content, like dialogue and instructions. The data section is a place that can contain custom-made attributes and elements.

The variable section includes global variables used throughout the game. An example is a variable that stores the values of multiple choice selections made by the user, to be used later by to assess the progress or performance of a user. The variable section also includes global variables used by the game.

The consequence section contains actions that are very much common to most games, like what happens when a non-player character is clicked on, or actions to access a menu. The consequence section probably could use a better name, and is rarely changed by the game programmer. Most actions are re-used between games.

The control section is where the finite state machine of the game is mapped out, including states, entry conditions, exit conditions, actions, things to execute and transitions. The control section makes reference to all the variables, resources and data defined in the other 4 sections. Essentially, the control section is the game. The game script makes use of indirection and scoping, and an advantage of this script is that its organization fosters code re-use, especially in the consequence section.

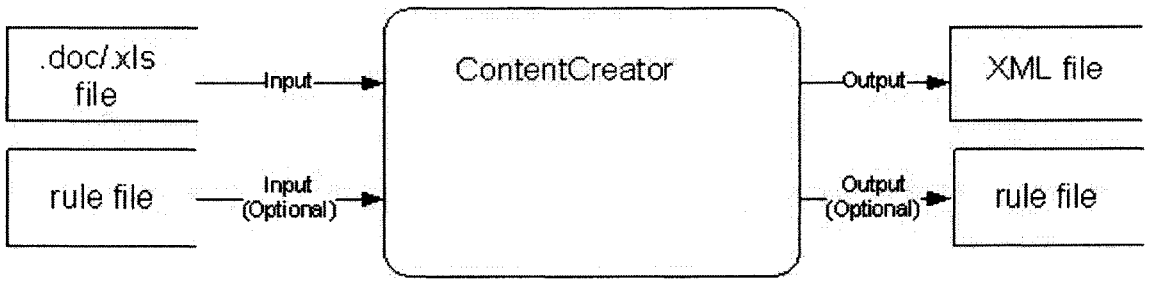

Figure 6.1: High-level dataflow diagram for converting files using *ContentCreator*

6.5 The Conversion Tools

The alpha names for the software tools are called *ContentCreator* and *Content-Converter*, respectively.

6.5.1 *ContentCreator*

The main purpose of the *ContentCreator* is to convert any *Word* or *Excel* file into XML. Through the interface, the user can open a *Word* or *Excel* file, designate a destination XML file, open a rule file and save a configuration of rules. Most of the user interface is devoted to defining the parsing 'rules' that the program will follow to convert the file. For example, the *ContentCreator* expects the document file to be organized by tables, styles or headings and allows the user to define the structure of the XML from the cells, style types or heading types. The user can preview the generated XML from within the program.

This tool is being used by the game development team to generate XML for character actions and narratives from tabulated *Word* files, created by a number of creative writers. Currently, the resulting XML code is copied and pasted into the game file, by the Game Programmer, at an appropriate time during the build process.

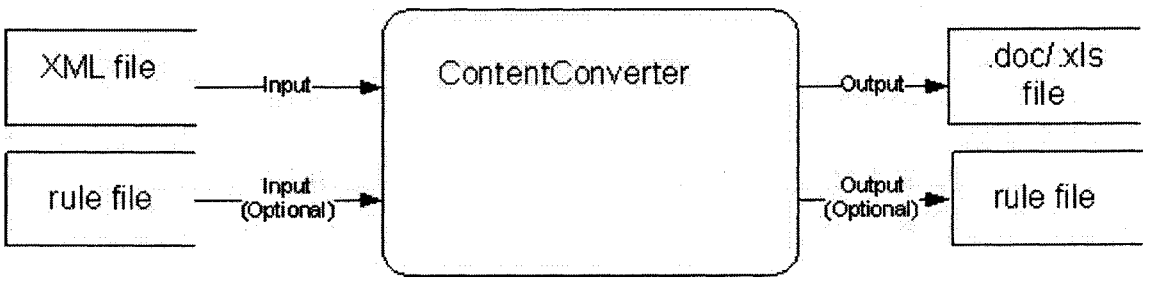

Figure 6.2: High-level dataflow diagram for converting files using *ContentConverter*

6.5.2 *ContentConverter*

This companion tool to *ContentCreator* converts XML back into *Word* or *Excel* documents. It allows users to ignore tags, put styles on tags and make fields read-only. The intent is to provide the flexibility to show parts of the game script in a format that is not XML. This tool is used by the game development team to convert XML game scripts into a readable format, in particular to show actions and narratives to customers and subject matter experts for editing and approval.

The members of the game design group admitted that this tool, *ContentConverter*, was easier to use than *ContentCreator* because knowledge of *Word* (or *Excel*) was expected by the program, as opposed to knowledge of XML. That is, *ContentConverter* reads in the XML and asks the user to define conversion rules in terms of commands, styles and terminology used in *Microsoft Word*.

As a result, we decided to concentrate on the more difficult of the two, *ContentCreator*. We surmised that defining XML rules were much more difficult and might require additional support.

6.6 Usability Study of the *ContentCreator*

6.6.1 Method

A usability study, or usability test, is a structured interview that focuses on specific features of a given interface or prototype to determine user understanding and to identify usability problems [58].

6.6.2 Subject and Workplace

Since our last visit, the development team was expanded to include a new member and role: a UX Specialist. She was available to act as a *subject* in this study. We will refer to her as the Story Integrator. She was primarily selected however because her role evolved to include the time-consuming tasks of converting and incorporating dialogue into game code using the internal tools.

Story Integrator Profile: This person works at GameCompany as a user experience (UX) designer and researcher. She works on serious games, web portals and authoring technology. She also runs usability studies on the games developed by the company. She is an expert *Word* user, a beginner with *Excel* and knows only the basics about XML. She also occasionally uses internal authoring and conversion tools. While she does have a UX background, the Story Integrator is the subject of this particular study not the interviewer.

Other members of the team included: Creative Writer, Game Programmer, Graphic Designer and Sound Designer.

6.6.3 Procedure

A preliminary interview was held with the Story Integrator to explain the purpose and procedure of the study and to gather background information, including a

description of her role at GameCompany, as presented in her profile. The interview was semi-structured with introductory questions that lead to deeper discussion.

Following the interview, the Story Integrator was asked to start *ContentCreator* and was asked to recall her initial impressions of the software, her current impressions and any overall thought about software, the layout and focus of attention.

She was then presented with two scenarios of use. The first involved loading a *Word* file, defining conversion rules and performing the conversion. Her goal was to create an XML file with the same format as a sample file. The second scenario involved saving the conversion rules and applying them to a second XML file.

The subject was asked to work-out-loud (or think-out-loud [62, 19]) to explain what she was thinking while she performed the tasks.

Scenario 1 Instructions

The instructions given to the participant, for the first scenario, are as follows:

Scenario 1: Your manager contacts you and requests your help in organizing the company library. To start, you are given a Word document, containing information about books, by your manager and asked to convert it into an XML file. Your manager is also kind enough to give you a sample of an XML file, showing you the desired format for the XML. Your manager also wants you to save the settings so that other Word documents containing book information can easily be converted in the future. You are given use ContentCreator to perform these tasks. You will find the Word file, called Books.doc, in the Scenario1 directory. Please save all your work in the same directory.

The desired organization, in words, is a library organized by subjects, with each subject containing books. Each book has the following elements: a type, title, and one or more authors. Subjects have an attribute called 'code'. Each book has an attribute called

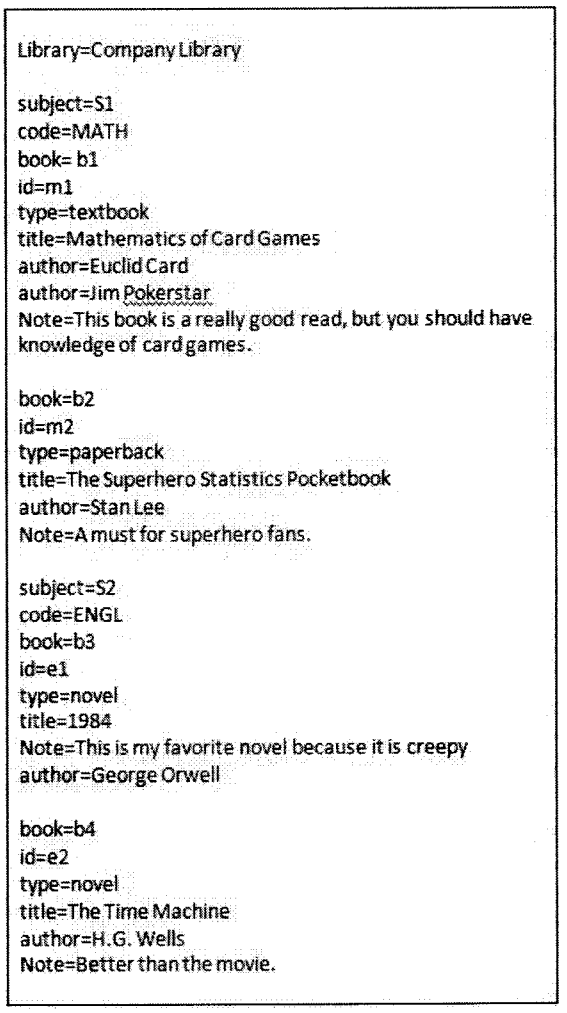

Figure 6.3: The text content of the Books.doc file used in Scenario 1

```
- <library>
  - <subject code="PHYS">
    - <book id="p1">
        <type>Textboook</type>
        <TITLE>Physics for Dummies</TITLE>
        <author>Bill Nye</author>
        <author>Isaac Newton</author>
      </book>
    - <book id="p2">
        <TITLE>The Thermodynamics of Cooking</TITLE>
        <type>Paperback</type>
        <author>George Foreman</author>
      </book>
    </subject>
  - <subject code="COMP">
    - <book id="c1">
        <type>Training Manual</type>
        <TITLE>Oracle 101</TITLE>
        <author>Mr. Delphi</author>
      </book>
    </subject>
  </library>
```

Figure 6.4: The sample XML file used in Scenario 1

'id'.

This scenario is divided into 4 tasks and multiple sub-tasks:

Task 1 - Load the Word File *Start the ContentCreator program Load the Word file called Books.doc*

Task 2 - Set up conversion rules *Set up the conversion rules. Feel free to explore the program, use intermediate steps, preview and save files until you are happy with the result.*

Task 3 - Save conversion rules *Once you are happy with the conversion rules, save the rules so that you may use them later with different files.*

Task 4 - Save the final XML file *Save the final XML file to the Scenario1 directory.*

Scenario 2 Instructions

The instructions given to the participant, for the second scenario, are as follows:

Scenario 2: Your manager sends you a second file, called NewBooks.doc, which is in the same format as the file in Scenario1. He wants you to convert this file to XML in the same format as in the previous scenario. You should use the rule file created step 3 of Scenario 1.

This scenario is divided into 3 tasks and multiple sub-tasks:

Task 1 - Load the Word File *Start the ContentCreator program Load the Word file called NewBooks.doc, located in the Scenario2 directory.*

Task 2 - Load the Rule File *Load the rule file created in Scenario 1. If you completed Scenario1 successfully, it should be located in Scenario1 directory.*

Task 3 - Convert and Save *Convert the file and save the xml file in the Scenario2 directory.*

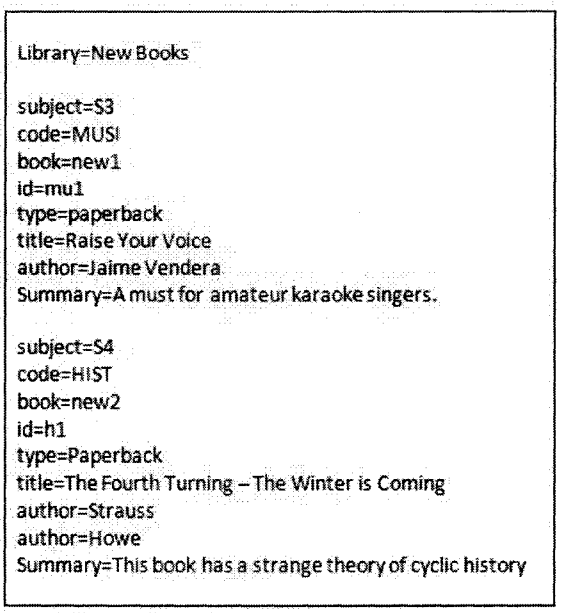

Figure 6.5: The text content of NewBooks.doc file used in Scenario 2

Task Performance Key

Each task performed by the subject will be summarized by the following two metrics, time to complete and correctness [58].

Time to Complete

0 Fail

1 Succeeds very slowly, in a roundabout way

2 Succeeds a little slowly

3 Succeeds quickly

Correctness

0 Fail

1 Many errors

2 Some errors

3 Succeeds quickly

6.6.4 Equipment

A laptop computer, with *Microsoft Office XP* and the *ContentCreator* installed on it, was used in the testing. The observation session and the resulting conversation were recorded using a digital audio recorder. Some notes were taken on paper.

6.6.5 Results

We spent two hours observing and talking with the Story Integrator while she performed a series of typical tasks using alpha version of *ContentCreator*. She spoke of her previous experience, including some anecdotes of painpoints encountered when using the program. While there were no major problems loading .doc files, loading rule files, viewing the created XML and saving rule files, her anecdotes were supported by observations during the rule creation process. She employed a trial-and-error strategy in an attempt to understand the behaviour of various options in rule creation, switching between views of the XML and the *ContentCreator* program. After six attempts, the resulting XML was only similar to the desired result. The program does have potential, but there are some interface problems that must be addressed before further usability testing and public consumption.

> Interviewer "So, what you're saying is: if you had a real-time preview pane, some in-program indication of what the rule options did, the non-essential information hidden, then you would think the tool would be more usable?"
>
> Story Integrator "Yes, I think that it would make a big difference. Hopefully then I could do the simple things myself".

6.6.6 Observations and Analysis

Scenario 1

Interview notes from Scenario 1 are available from the author by request.

Task 1- Load Word File in Program

User: Story Integrator

Time to Complete: 3 (Succeeded Quickly)

Errors: 3 (No errors)

Summary: Opens the *Word* document first to see what type of conversion to perform, find the delimiting character and check for whitespace. Once this is done, she quickly and successfully loaded the *Word* file in *ContentCreator*.

Observations and Notes:

- She usually opens the Word document to:
  - Check for whitespace, as this is known to cause problems or errors with the conversion tool
  - Look for 'delimiter' character and finds that the file was using '=', so you know what kind of conversion to perform (i.e. tabled, styled or labeled).
- Says she is more comfortable with tables, as that is what they are currently using.
- She notices that the document file contains extra information than the desired format requires
- She loads the document in ContentCreator without problem

Task 2 - Set up conversion rules

User: Story Integrator

Time to Complete: 1 (Created a series of rules in a very slow, roundabout way)

Errors: 1 (Many errors)

Observations and Notes: After six attempts, the user creates an XML file with a similar, but not exact, format as desired. She used a trial-and-error approach, starting first with her best guess and then looking at the default output. She then, hesitantly, made a few changes in the rules, previewing after each, to try to learn the behaviour of grouping, levels and other rule options.

She expressed a desire to see immediate feedback on what each option and check-box in the rule creation area did. Switching tabs to view the resulting XML between each incremental change to the rules, however, did not result in the desired understanding. Alternatively, she suggested that in-program examples or descriptions for each part of the rule creation.

During this process, the user encountered an error and lost patience, requiring prompting to continue. The end result (Figure 6.6) was an XML file that was close to, but not in, the desired format (Figure 6.4).

Task 3- Save conversion rules for later use

User: Story Integrator

Time to Complete: 3 (Succeeded quickly)

Errors: 3 (No errors)

Observations and Notes:

There were no problems encountered with saving a rule file. This task was done quickly.

Task 4- Convert and Save the File

User: Story Integrator

Time to Complete: 3 (Succeeded quickly)

Errors: 3 (No errors)

Observations and Notes: There were no problems encountered with converting and

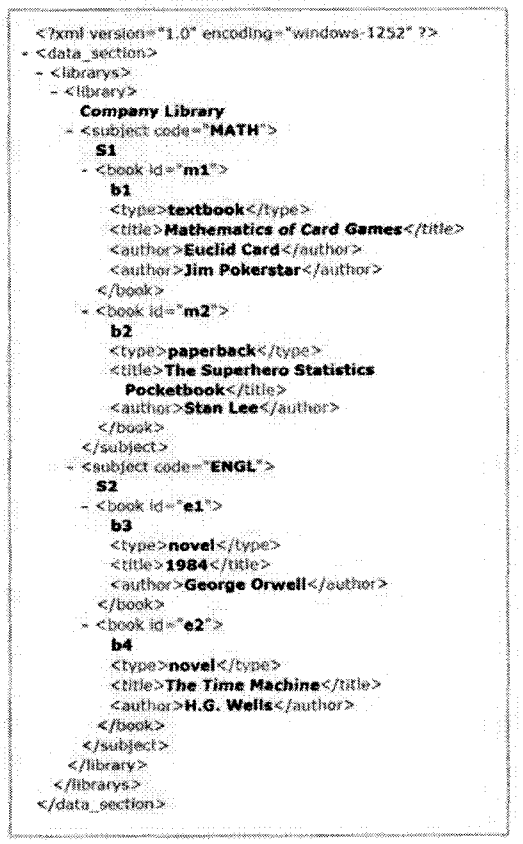

Figure 6.6: Final User Result of Scenario 1: Task 2

saving the file. The subject recalls that when she converts many files one after another, she will often forget to change the destination file. Current behaviour is that the destination file text box remains filled when a new file is loaded. It should be empty when a new file is loaded.

Scenario 2

Task 1- Load Word File in Program

User: Story Integrator

Time to Complete: 3 (Succeeded Quickly)

Errors: 3 (No errors)

Summary: Opens the *Word* file without any problems. She does comment that the program gives little indication that it is doing anything.

Notes and Observations:

- During the loading of a *Word* document, the status of the program was only visible right before the program became responsive, but after a long delay.

Task 2 - Load the Rule File

User: Story Integrator

Time to Complete: 3 (Succeeded Quickly)

Errors: 3 (No errors)

Summary: Opens the rule file without any problems. She does comment that the behaviour of clicking on the empty text box to get a file loading prompt is a little strange.

Notes and Observations:

Story Integrator: "To me, it doesn't seem obvious that you should be able to click on that".

Task 3 - Convert and Save

User: Story Integrator

Time to Complete: 3 (Succeeded Quickly)

Errors: 3 (No errors)

Summary: The file is automatically saved when you click on the Convert XML link.

Notes and Observations:

- Program needs better indication that the act of converting the XML is actually saving the file.
- Interviewer: "Did you save it?".
- Story Integrator: "It's automatically saved, right?"

6.6.7 Interpretation Notes

- The Story Integrator desired support that includes a real-time preview of the XML being created as the rule options are selected. The user would like to see this to the left of the rule selection area. The constant context switching for even a simple XML file suggests that this information should be present on the same screen as the rule options. This could be described in terms of HIDDEN DEPENDENCIES and VISIBILITY. There are hidden links between the rule options and the resulting XML and the resulting XML is not visible from the rule creation screen.
- Alternatively, an in-program indication of what each rule option means may offer a quick fix. Roll-over tool tips or pop-up links, containing definitions and examples are an acceptable possibility.
- The entire paradigm suggests some knowledge of XML is required.

- A re-work the interface for 'grouping' options is required. Changes made by the program to other rows of rules must be clearly highlighted and visible to the user. Grouping attributes and grouping XML Tags are two different things, but are both represented by a checkbox, constituting a high degree of hidden dependency.

- Non-essential information should be removed or hidden. This includes most of the left hand links. In addition, most of the left hand links are inconsistently named with similar functionality elsewhere (i.e. QuickView tab vs. View XML link). The View XML link is 'hidden' amongst the other rarely used links.

- The destination file textbox remains populated when a new Word file is loaded. This should be at least blanked out or placed near the View XML button, or perhaps changed to a Save XML button. As is, the user may accidentally save over top of a previously converted file.

- Load times for document files are extremely long (3 minutes, 30 seconds for a 30 page file). Conversion times are very short in comparison (a few seconds). The program has poor indications that the conversion is complete.

- The View XML link should be a button, located in lower, right-hand side of the screen.

- Change the orange color to something more neutral.

- While not encountered during the session, the program's finicky handling of whitespace was problematic during the cognitive walkthrough and the dry run for the usability testing.

6.7 Discussion

6.7.1 Usability of Conversion Tools

The lack of support and visualization during rule creation process was the most problematic issue identified. In general, this is the very feature set that makes this program a valuable tool. From the observations, the program needs to provide the information that the user needs to make decisions when building the rules, perhaps in the form of hover tips and examples. The work practice of the Story Integrator suggests that a real-time view of the XML being created would be much more favorable and helpful to the user.

We see two conflicting requirements for *ContentCreator*. First, the tool is meant to be a *generic* conversion tool, usable for anyone including the game developers at GameCompany. Unfortunately, the game developers at GameCompany might be better served by a tool that automatically converts documents to parts of the desired game XML.

To illustrate, here is a conversation held with the Story Integrator at the end of the usability testing when she was asked about the process of converting files:

> Interviewer: What do you think about this process?
>
> Story Integrator: I think I'm unnecessary. We should provide the SME and writers templates which, once filled in, are then automatically converted and entered into the game.
>
> Interviewer: How long does it take to format a Word file when it is received by the customer?
>
> Story Integrator: A long time. Sometimes hours and hours. Near the end of the game lifecycle, this can take the whole day: formatting, converting, making sure that [the Game Programmer] has the right version.

The Story Integrator would receive every document file that was passed between the subject matter experts, learning specialists and creative writers. Between each pass, the Story Integrator would check the format of the document file, correct any formatting errors and convert the file.

To compensate for the difficulty with setting up rule files using the conversion tool, the document files were formatted using tables. The right commenting, whitespace and styles are all required to be a certain way for the established rule file to convert the document properly. The structure of the tables within the document file ended up describing the hierarchical format of the XML desired.

> Story Integrator: We're lucky our SMEs are really patient. We give them explicit instructions to only fill out specific cells, not to add whitespace and to use the comment functionality in Word.

Essentially, the work process has evolved into using templates and not changing the rule files. Aside from input from the content creators and the subject matter experts, other information populates these tables, like scores and identification tags that link these tables to game code.

> Story Integrator: If this [information] isn't put into the Word document, it gets lost throughout the iterations.

When asked how they keep this extra information from being edited or erased, she replied:

> Story Integrator: We now use read-only cells and invisible cells in the tables

The usability test clearly indicated ways in which the interface for the conversion tool may be improved. It also indicated that exposing an interface for such conversion tasks in the process of game scripting may not be the most beneficial solution.

6.7.2 A Cognitive Ethnographical Study

In a related study [100], Tran performed a cognitive ethnography [108], observing the development team at GameCompany for a week, capturing 30 hours of audio recordings of the team's conversations during the tail-end of the development cycle of a serious game. These observations were also made one year following the contextual inquiry (see Chapter 5). The layout of their workspace was such that all members of the development team could see and hear each other from their individual work stations. Tran sought to describe the team, their environment and work processes using the Distributed Cognition [49] theoretical framework, Activity Theory [66] and Cognitive Dimensions of Notations.

From Tran's raw data, we notice that [100]:

- there was one conversation between the Story Integrator and the software developer that designed the conversion tools. The Story Integrator sought help to use the tool.
- adding dialogue to the XML code is an iterative & incremental process (and common) process
- the re-design of older games is a common task
- the design current process is incremental and iterative
- They use *Subversion* to track changes and versions of a game during development cycle

And that some of Tran's observations directly relate to those key concerns identified by the contextual inquiry (see Table 6.1).

Key Concern Identified in Contextual Inquiry (see Chapter 5)	**Observations 1 Year Later**
Communication between narrative in document files and the XML code is disjointed	the conversion tools are in alpha stage
Adding dialogue to the XML code is an incremental and common task	the conversion tools are in alpha stage
Debugging is time-consuming and a common task. An organized approach and support is needed.	bug bashing is a common task
Debugging is time-consuming and a common task. An organized approach and support is needed.	the development team now uses a bug tracking system, called *Mantis*, to track, manage and report bugs and usability issues.
Debugging is time-consuming and a common task. An organized approach and support is needed.	in 30 hours of recorded work, the development team collectively made reference to: *Mantis* 18 times, Network resources, such as shared drives, 37 times.
Use of 3rd party tools is highly prevalent	oXygen is still used to edit game XML
oXygen provides no support in matching tags in the game code to the game resources, such as sound or art files.	the conversion tools are in alpha stage
Current 3rd party tools do some of the things that are necessary to get the job done.	they are in the initial design phase for other tools to support the programmer

Balancing creativity and fun with learning requirements is difficult	the Creative Director is no longer at the company; creative writing tasks are now divided amongst several individuals, labeled Creative Writers

Table 6.1: Key concerns mapped to observations 1 year later at GameCompany

Their development process has evolved from purely an iterative process to include incremental steps, tracked by versions, which may or may not be tested and evaluated after each increment. User testing for game dialogue and evaluation by SMEs and customers most often results in iterative changes. It is not trivial to note that it is these iterative changes, involving dialogue changes, to the game code remain time-consuming and difficult to perform, even with the alpha conversion tools, while the introduction of bug-tracking systems and version controls have streamlined other parts of their overall process.

6.8 Conclusion

The GameCompany realized the need for support tools for creating and updating game content, in particular dialogue. Unfortunately, the use of generic tools for converting *Word* documents to XML (and back) proves somewhat problematic with respect to game scripting. There is still a requirement to understand XML code and the structure of the game code and to map the narrative structure to the game XML. This provides evidence that a domain-specific language and support, with automatic mappings to game code, is more suitable (see Advantages of Domain Specific Languages in Section 2.7). The game programmers and the content creators want to visualize the structure of the game and have an easy way of updating the narrative.

In the next chapter, we will present what we learned is needed to support narrative scripting.

Chapter 7

Conclusions

7.1 Summary

In this thesis we have addressed the issue of support for scripting interactive narratives for serious games.

In Chapter 3, we examined interactive narrative systems and techniques revealing the support required for interactive narrative scripting. In particular, the focus was on supporting non-programmers and, from the results, we proposed the basis of a pattern language for support.

In Chapter 4, we described how we used Cognitive Dimensions of Notations framework to conduct a study of a commercial toolset for game creation. We identified weaknesses of an interactive narrative system imbedded within an existing game and the supporting authoring tool. In particular, the commercial toolset lacked a meaningful way of showing and organizing story structure and presenting story-related variables. Instead, the user was forced to keep track of these things across file structures, modules and inside of hidden programmatic scripts.

In Chapter 5, we described how we used principles of Contextual Design to conduct a study that observed both the game programmer and the creative writer as they did their work at a professional serious game company. We showed that, in practice, scripting a serious game is an iterative process and that this process relies heavily upon the input of specialists to produce the game. In particular, the creative writer relied heavily upon the game programmer to integrate story content into the game.

This type of tag-team seemed to be the assumed norm in professional game creation, both in practice and in the literature.

In Chapter 6, we illustrated how integration tools need to relate to the domain. We conducted studies, performing both a cognitive walkthrough and a usability study, to examine the tools used for integration at GameCompany. We found that the use of generic third-party tools and generic internal tools were still not sufficient in supporting rapid and seamless integration of narrative content into the games. Integration remained a hands-on and time-consuming task for the professionals involved. It also still required a specialist who was not the creative writer. The frequency of integration made this a major pain-point.

Overall, we studied the topic of interactive narratives from four perspectives: 1) the literature, 2) IN tools and a commercial game development tool, 3) professionals at work and 4) the tools they developed to support their work. From each study, we came to broadly similar conclusions.

7.2 A model for supporting serious game scripting

Based on our research, we present our model of support for scripting interactive narratives of serious games:

1. Provide Authors with a tool that primarily supports interactive narratives

2. Support Story Building and Visualization in the Tool

3. Support frequent updates and testing of the Story during Creation

4. Support automatic integration of the narrative

7.2.1 Provide Authors with an interactive narrative tool

The patterns in Chapter 3 describe what authors need to create interactive narratives. First off, they either need a game system that has an authoring tool that is an interactive narrative system, like *Inform 7*, or that supports interactive narratives within a game, much like what the *Neverwinter Nights Aurora Toolset* attempts to support. In the latter system, the author can leverage the game resources, map creators and graphic elements to create a game and both dialogue and journal editors to create the interactive narrative components. Within the tool itself, a creative writer needs to be able to access, organize, and share both default and custom game resources. A creative writier also needs to work with terminology and concepts that are closer to those used in stories and creative writing than that of programming languages.

7.2.2 Support Story Building and Visualization in the Tool

While stories have structure, creative writers need the structure to be part of the creative process and not forced upon them. The structure of the story needs to be highly visible and not buried across files, resources and hidden variables, as seen in Chapter 4. With respect to the dialogue in the game, the creative writer at GameCompany appreciated the freeform nature of a WYSIWYG editor (see Chapter 5). The act of creative writing is dynamic. In other words, the creative writer goes through many drafts that may include ancilliary material. While writing a particular draft, the writer types and erases often. Freeform support addresses the dynamic nature of writing interactive narratives and the creative writers are used to WYSIWYG editors like *Word*. However, as the narrative evolved during design and development, it took the form of a structured document that used tables and headings to capture chapters, characters and descriptions. The structure was used to

allow for conversion into game code (see Chapter 6).

An interactive narrative authoring tool should support the Creative Writer by providing a freeform editor that allows dialogue and story structures to be added, edited and linked together in any order. The dialogue editor itself should follow a WYSIWYG paradigm.

The author should be provided a view of the overall narrative structure and, from that view, should be able understand and walk through the many story paths in of a branching story lines.

7.2.3 Support frequent updates and testing

In chapters 3 through 6, we illustrate the need to allow the story writer to test and play through the story and parts of the story throughout the scripting process. The story writer should not have to play through the entire story from the beginning to test a particular part of the story. Initially, we captured the need for this type of support in the pattern Read it back to me but it was also encountered in the study of the *Aurora Toolset* and during our visits to GameCompany. The ability to play through the narrative at any point during development is required to support the narrative development itself, with the added benefit of catching bugs earlier on in the development process, thus reducing overall cost of development.

A narrative scripting tool should provide the creative writer with the ability to play-test a part of the interactive story at a user-defined starting-point through to a desired end-point. If the interactive narrative system is embedded, like in the *Aurora Toolset*, then this support could take the form of a lightweight play-test, whereby only the narrative components could be stepped through and previewed, independent of loading graphics and virtual worlds. This would allow for frequent iterations of scripting and testing, while reducing required system resources required to play-test.

In addition, this approach would remove the need to integrate in order to play-test.

7.2.4 Support automatic integration of the narrative

Integration support is essential to removing the creative writer's dependency on the game programmer, as we encountered in the literature and illustrated in Chapter 5 and Chapter 6.

The *Word* documents that were converted into XML using the conversion tools, did contain a tabular structure in which the dialogue was organized. However, this structure was more of a construct to make the conversion process, using the conversion tools, easier. We believe that if the narrative content took a structure of a story, the structure would feel more natural to the creative writer. Ideally, the integration support would map the story structure and narrative content to the underlying game logic and code. No special formatting and conversion would need to be performed by the game programmer or integration specialist, as it would be automated by the authoring tool.

7.3 Our Contributions

Given the unique requirements and goals of serious games and their potential growth and ability to support persuasion and learning, we set out to understand the support required to script serious games, in particular interactive narratives for serious games.

To this end, we performed the following work:

- We conducted research in narrative scripting tools and techniques and presented our findings in the form of patterns, organized in a pattern language, for support of game scripting.

- We employed and combined two techniques in HCI to conduct a study of a

commercial game development tool that contained an embedded interactive narrative system. Our contributions included a novel way of codifying the interpretation notes. We found that the presentation and visualization of story structure was poor throughout the tool, even though it was specific to the domain of game creation.

- We employed a third technique in HCI to conduct a study of professional game developers of serious games. We observed that game development was a highly iterative process and that creative writers relied heavily upon game programmers to integrate dialogue and stories into the game. Generic third-party tools did not support easy play testing and integration.

- Two more HCI techniques were utilized to examine the tools developed by and for professional game developers to automate integration tasks a year after the first study at GameCompany. We found that the generic design of the tool still required the knowledge of specialists for integration.

Together, what we learned in the studies form a model of support for scripting interactive narratives of serious games, to be used as a guide for future research.

In addition to conducting the above research, we supported the general research efforts towards the ORNEC-funded project entitled End-User Scripting for Online Training by interacting often with members of GameCompany, presented our initial findings to employees GameCompany and aided in the codification and analysis of Tran's ethnographic study.

7.4 Future Work

Our emphasis in this thesis was to employ HCI techniques to explore what is necessary to support scripting of interactive narratives for seriuos games. We suggest that the

most important next steps in the research involve investigation and work in two areas, as described below.

7.4.1 Map to an Underlying Model

To achieve the automatic code generation and the play-test anytime requirements, we will have to map the story structures directly to state machines or Petri Nets. In other words, in the least, we need an underlying model that can represent states, transitions and actions. We also might want to consider variants like PN-model or concurrent hierarchical state machines (CHSM) [59] for hierarchical organization and concurrency. A tool might allow for output to multiple models, depending on the story structures created by the user, or to a custom code structure, like the proprietary game files of GameCompany.

7.4.2 Design Story-Specific Conversion Tools

Another area of future work is to create a conversion tool that is story-specific, that allows for virtual preview of the interactive narrative and that automates conversion into XML. This tool could take the form of a Macro-enabled *Word* document to leverage the WYSIWYG user interface that is preferred by the creative writers. We saw in Chapter 5 that creative writers want to work with text in an unstructured way and iterate to a game structure.

Qualitative user-testing on low-fi prototypes and mock-ups should be performed to ensure that the tool resonates with would-be story writers. The creative process is subtle and highly iterative and the story writers are so heavily invested in the craft of creative writing that we want to take special care by creating mock-ups and gathering feedback. Ideally, this user research would be followed by a high-fi prototype that consisted of a UI that allows stories with simple story elements to be created. Ideally,

subjects for these studies would include non-programmers. They might also include GameCompany's clients, potential clients and those whom are the subject matter experts in government and industry, teachers and students from middle schools to universities.

Appendix A

Observations from Cognitive Dimensions Analysis of the *Aurora Toolset*

Below is a table showing the summarized, uncoded, results of the Cognitive Dimensions analysis of the Aurora Toolset. The table is ordered by dimension.

Dimension	**Dimension Level**	**Issue Description or Thumbnail**
CLOSENESS OF MAPPING	Low	Meaning of existing behaviour scripts is unknown
CLOSENESS OF MAPPING	Low	Must learn to program scripts to describe new behaviours or edit existing ones
CLOSENESS OF MAPPING	Low	Must learn to program scripts to make non-simple conversations in the game
CONSISTENCY	Low	Must explicitly add blueprints when exporting an area, but instance in the area are automatically added
CONSISTENCY	Low	Undo command does not work for all tasks
CONSISTENCY	Low	Eraser tool does not behave the same for all tasks
HIDDEN DEPENDENCIES	High	Triggers for journal updates are not seen in the Journal Editor
HIDDEN DEPENDENCIES	High	Links exist between module scripts and conversation, but are buried in conversation trees

HIDDEN DEPENDENCIES	High	Links exist between module scripts and conversation, but are not visible in the Module Content Pane
HIDDEN DEPENDENCIES	High	System displays no concept date created or time last edited for resources
PREMATURE COMMITMENT	High	Must create area before content
PREMATURE COMMITMENT	High	Must create a module before content
PREMATURE COMMITMENT	High	Must define area size before creating content
PREMATURE COMMITMENT	High	Must select tile set for an area without being able to preview tiles
PROGRESSIVE EVALUATION	Low	Can check work, but only from the beginning
SECONDARY NOTATION	High	Comments about an exported resource are only available at end of import process
VISCOSITY	High	Resizing the Map contains many overly complicated steps
VISCOSITY	High	Many tasks involved in moving resources between modules
VISIBILITY	Low	Cannot view resources inside exported file without importing
VISIBILITY	Low	Poor visual cues to differentiate between the edit screen of an instance and a blueprint
VISIBILITY	Low	Local variables are not visible or accessible from the main interface

VISIBILITY	Low	Script preview pane is too small. Cannot resize and must open up editor to view in larger window

Table A.1: Uncoded Observational Notes

Appendix B

Full Affinity Diagram

Below is the affinity diagram that includes all the interpretation notes from which it was inductively built. These notes consist of the important observations made during the contextual inquiries with employees at an Ottawa-area gaming company.

- Red: A Highly Technical Activity
  - Orange: Programming Is Involved
    - Pink: I Don't Like Documentation
      - Game_Programmer-14 Revision notes are not used
      - Game_Programmer-15 Does not document small changes in components
      - Game_Programmer-21 Lacks incentive to document work
    - Pink: I Debug
      - Game_Programmer-53 Adds comment output to game to test if component is loading properly
      - Game_Programmer-33 Debugs in Flash
      - Game_Programmer-47 Uses action script comments to help debug
      - Game_Programmer-32 Tests to see if components are properly integrated by compiling and running
      - Game_Programmer-30 Debugs XML components after it is integrated
      - Game_Programmer-46 Flash output window displays too much irrelevant output

- Orange: The Work Needs To Be Visualized
  - Pink: Computers Are Not Always Used
    - Creative_Director-53 Prefers hardcopies because it is easier to flip through paper than to scroll electronic files
    - Creative_Director-54 Softcopy (i.e. typing) usually saved for final edits only
    - Creative_Director-08 Uses whiteboard to write training manual titles to keep track of what needs to be covered
    - Creative_Director-36 Hard copy is printed along soft copy
    - Game_Programmer-07 Keeps hard copies of documentation at desk
    - Game_Programmer-08 Creates low fidelity prototype on paper or whiteboard
    - Game_Programmer-43 Game overview displayed in black book, whiteboard and memory
    - Game_Programmer-72 Draws FSM in notebook
  - Pink: The Game Is Layered With Respect To Components And Graphics
    - Game_Programmer-58 There is no view of component depths
    - Game_Programmer-59 Scroller depth must be below other interaction components
    - DI Game_Programmer-51 Resources depth should be more visible in code view
    - Game_Programmer-60 Depths are set by a semi-arbitrary system
    - Game_Programmer-61 Checking if depth is right requires playing the game
  - Pink: I Want A High Level View Of My Work
    - DI Creative_Director-30 Allow section of story that are committed

to be views separately

- Game_Programmer-45 Uses xml comments to separate group in xml
- Game_Programmer-44 Oxygen does not put text of XML comments in tree view
- Game_Programmer-42 Single monitor used solely for code development (said he'd like two monitors)

– Orange: Existing Tools Have Some Of The Desired Features But Some Tasks Could Be Automated

* Pink: Existing Tools Do Some Of The Things I Need To Do My Work
  - Game_Programmer-02 He uses flash and oxygen simultaneously
  - Game_Programmer-02 He uses flash and oxygen simultaneously
  - Game_Programmer-12 Edits XML in Oxygen
  - Game_Programmer-04 Views documents in Word
  - Creative_Director- 32 Edits existing stories in Word (not in XML)
  - Game_Programmer-11 Edits action script in Flash Developer
  - Game_Programmer-62 Uses OpenOffice to view documents
  - Game_Programmer-01 He integrates graphics, narratives and game logic
  - Game_Programmer-18 He prefers oxygen features
  - Game_Programmer-57 He works between Flash and Oxygen
  - Creative_Director-14 Scenario Vaults are ideally created in Word because of unlimited spacing and easy formatting
  - Game_Programmer-75 He likes Oxygen (tree view, dbl click on node to expand, copy/paste nodes, attribute preview in tree, auto complete, knows XML structure)
  - Creative_Director-01 She works with MS word
  - Game_Programmer-09 Create high fidelity prototype in flash

- Creative_Director-25 Creative_Director likes MS Words for editing, WYSIWYG, spell-check, unlimited space
- Game_Programmer-52 Oxygen learns XML structure; saves typing

* Pink: I Want To Automate Some Tasks
  - DI Game_Programmer-80 Tool can translate story narrative into XML
  - DI Game_Programmer-81 FSM diagram needs a way to be validated
  - Game_Programmer-73 FSM diagram gets translated into XML
  - DI Game_Programmer-74 Tool to translate FSM diagram into XML
  - Creative_Director-35 Wants to add narrative directly into game herself

– Orange: Tools Should Help Component Re-Use

* Pink: I Want Tool To Be Aware Of Resources And Components
  - DI Game_Programmer-67 Program does not know which section needs editing
  - DI Game_Programmer-68 An "insert component" feature would help set configuration and set IDs
  - DI Game_Programmer-69 Tool should know games assets (components, graphics, and code)
  - Game_Programmer-66 Adding new components requires edits to core XML in several places
  - DI Game_Programmer-20 Wants XML editor to know component XML structure
  - Game_Programmer-54 Looks up function names in other files then copies into core XML

- DI Game_Programmer-19 Wants Oxygen to memorize config options
- DI Game_Programmer-49 Tool should force unique values for certain attributes that require unique values
- Game_Programmer-17 Views each component before using
- Game_Programmer-31 Copy and pasts XML components into core XML
- Game_Programmer-55 Oxygen does not know available components
- Game_Programmer-48 Action script comments must be turned off manually
- Game_Programmer-56 Oxygen does not know component configuration

* Pink: I Want to Re-Use Components
  - Game_Programmer-76 Distil has 12 components
  - Creative_Director-26 Creative_Director uses common layout for story skeleton
  - Game_Programmer-13 Reuses game component
  - Game_Programmer-78 Code can be attached to graphics
  - Game_Programmer-29 Modifies component XML to fit into core game
  - Game_Programmer-23 Uses components created by others
  - Game_Programmer-22 Uses test XML to see how XML works
  - Game_Programmer-28 Opens component code to see how component is loaded
  - Game_Programmer-65 Takes graphics from a main Flash graphic file, copies them into new flash
  - Game_Programmer- 50 Checks all associated sections of core XML

when component is added

* Pink: The Type Of Game Determines The Structure Of The Game
  - Creative_Director-42 Essential games are structured by opportunity, options and consequences
  - Creative_Director-43 Response ready type games are a "matching game"
  - Creative_Director-44 Interaction Assessor is question and answer
  - DI Creative_Director-31 Tool should allow different / news types of games
  - Creative_Director-33 Different game methods require different narrative structures

- Red: Teamwork
  - Orange: Development Is A Team Process
    * Pink: Communication is Informal
      - Game_Programmer-71 Continual communication about game direction throughout development
      - Game_Programmer-63 Talks to Steph about game world components and graphics
      - Creative_Director-48 Art directions is created in same process as narrative (they are created together)
      - Creative_Director-29 Communicates with artists informally (email, face-to-face)
      - Game_Programmer-05 Rarely uses design document
      - Creative_Director-49 Emails can be used to send sections of narratives
    * Pink: We Work On Games Simultaneously

        - Creative_Director-28 Rough drafts are used by artists and programmers to get a head start on development
        - Game_Programmer-25 Uses placeholder art if none is available
        - Game_Programmer-26 May integrate flask components before narrative and art is available
    * Pink: We Work Collaboratively On All Aspects Of A Game
        - Creative_Director-39 U04 - Product Manager marks up hard copy with comments requirements/customer requests
        - Creative_Director-58 Art direction is collaborated with others
        - Creative_Director-27 U04 - Product Manager and U03 - DESIGNER contribute to story skeleton with requirements and assessment weights
        - Creative_Director-24 Narrative is reviewed to see if they meet learning requirements
    * Pink: People Check My Work
        - Creative_Director-41 All edits reviewed by U03 - DESIGNER
        - Creative_Director-37 Annotations/comments are marked on hard copies
        - Creative_Director-40 Has to make sense of other people comments
- Orange: We Share Our Files
    * Pink: We Organize Our Files
        - Game_Programmer-39 Has expectation of where files should be
        - Creative_Director-45 1 electronic file for each story
        - Game_Programmer-36 Subdirectories are organized according to game structure / levels
        - Game_Programmer-34 Resources in wrong directory will produce errors
        - Game_Programmer-27 Create swf file and places them into game

(file) directories

- Game_Programmer-35 Component files are organized into subdirectories
- Game_Programmer-03 He retrieves documentation from shared network folder
- Creative_Director-50 Older games do not have one single file with all narratives
- Game_Programmer-37 Browses for missing files manually
- Creative_Director-51 Narrative text can be in various files (emails, MS Word and XML)

* Pink: We Keep Track of Changes
  - Creative_Director-52 Highlighted text means final versions
  - Game_Programmer-79 Graphics should not be modified if it contains code

* Pink: I Keep Track of Characters
  - Creative_Director-20 Prefers names and titles are not duplicated across and within games
  - DI Creative_Director-21 Database of used names and titles
  - DI Creative_Director-22 Sort characters names by ethnicity, popularity, and date
  - Game_Programmer-40 Black book keeps record of characters by game level
  - Game_Programmer-41 Uses game design to know about characters and their locations

– Orange: Team Members Have Different Roles

* Pink: Some Work Is Done In Order
  - Creative_Director-16 Creates story skeleton from scenario vault

and narrative summary

- Creative_Director-17 Story skeleton includes details about "opportunities" (interactive hotspots)
- Game_Programmer-77 Game structure created from story narrative document
- Game_Programmer-24 Gets narratives from Creative_Director

* Pink: Work Is Divided
  - Creative_Director-02 Edits story narrative
  - Game_Programmer-10 Retrieves final prototype from shared drive
  - Game_Programmer-16 Writes components

- Red: Strategic Creativity (Balance between Requirements, Learning and Fun)
  - Orange: Every Game Has Essential Requirements
    * Pink: I Need To Know The Requirements For the Game
      - Creative_Director-12 Determines what needs to be learned (breadth and depth of content)
      - Creative_Director-38 U03 - DESIGNER marks up hardcopy with comments for scoring/requirements
      - Creative_Director-15 Creates company profiles to determine content (knowledge about company determines how material is conveyed)
      - Game_Programmer-06 He knows design specification from face-to-face meetings
      - Game_Programmer-70 Decides if new components need to be created
    * Pink: I Need To Know the Subject Matter
      - Creative_Director-10 Talks to SME's by telephone to get training scenarios, policies and real-world examples

- Creative_Director-11 SME and Creative_Director discuss narrative summary
- Creative_Director-13 SME input can be used to create scenario
- Creative_Director-03 Learns about target audience from U04 - Product Manager via face-to-face meeting or documentation
- Creative_Director-04 Becomes familiar with training material

– Orange: Creativity Is Important

* Pink: I Create Characters
  - Creative_Director-55 She decides where NPCs are placed in game
  - Creative_Director-56 She decides what NPCs look like
  - Creative_Director-57 She decides NPC behavior/demeanor
  - Creative_Director-18 Determines names by browsing internet (school class lists)
  - Creative_Director-19 Determines job titles by browsing internet
* Pink: I Sketch and Save My Ideas
  - Creative_Director-46 Unused narrative are not deleted
  - Creative_Director-47 Unused narratives are copied to a "scrap file" (electronic)
  - Creative_Director-05 Dumps ideas into Word documents with short-hand
  - Creative_Director-06 'Scenario Vault' is a collection of potential game scenarios (like an electronic sketch pad)
  - Creative_Director-07 Visually scanning scenario vault helps inspire ideas or refreshes old ideas
* Pink: I Need To Be Creative
  - Creative_Director-23 Does not use checklist to satisfy requirements. Checklists are not "organic"
  - Creative_Director-34 She tries to add story elements into the game

so that it isn't just e-learning

- Creative_Director-09 'Strategic creativity' is key; being creative, yet satisfying learning requirements

Bibliography

[1] E. J. Aarseth. *Cybertext: Perspectives on Ergodic Literature.* John Hopkins University Press, London, 1997.

[2] J. Aldred, R. Biddle, C. Eaket, B. Greenspan, D. Mastey, M. Q. Tran, and J. Whitson. Playscripts a new method for analyzing game design and play. In *Future Play '07: Proceedings of the 2007 conference on Future Play*, pages 205–208, New York, NY, USA, 2007. ACM.

[3] C. Alexander. *A Pattern Language.* Oxford Univeristy Press, New York, 1977.

[4] E. Alhadeff. Microsoft shaping the serious games movement into a multi-billion dollar market, future-making serious games: The best of serious games that challenge us to play at building a better future. `http://elianealhadeff.blogspot.com/2007/11/microsoft-shaping-serious-games.html`, January 1, 2007.

[5] E. Alhadeff. Reconciling serious games market size different estimates. `http://www.businessandgames.com/blog/2008/04/reconciling_serious_games_mark.html`, April 2008.

[6] C. Ashmore. Storytron (review). Icosilune, `http://www.icosilune.com/2009/01/storytron/`, 2009.

[7] D. Balas, C. Brom, A. Abonyi, and J. Gemrot. Hierarchical petri nets for story plots featuring virtual humans. In *AIIDE*, 2008.

[8] C. Bateman. *Game Writing: Narrative Skills for Videogames (Charles River Media Game Development (Paperback)).* Charles River Media, Inc., Rockland, MA, USA, 2006.

[9] H. Beyer and K. Holtzblatt. *Contextual design: Defining customer-centered systems.* Academic Press, 1998.

[10] BioWare. Neverwinter nights. Infogrames/Atari Inc., 2002.

[11] BioWare. Star Wars: Knights of the Old Republic. LucasArts, 2003.

[12] A. Blackwell. Cognitive dimensions of notations: Design tools for cognitive technology. *Lecture Notes in Computer Science*, 2117:325–341, 2001.

[13] L. Blackwell, B. von Konsky, and M. Robey. Petri net script: a visual language for describing action, behaviour and plot. In *ACSC '01: Proceedings of the 24th*

Australasian conference on Computer science, pages 29–37, Washington, DC, USA, 2001. IEEE Computer Society.

[14] Blizzard. Press release: World of warcraft reaches new milestone: 10 million subscribers. `http://eu.blizzard.com/en/press/080122.html`, January 22, 2008.

[15] P. Bockman. As larp grows up. `http://eu.blizzard.com/en/press/080122.html`, January 22, 2008.

[16] I. Bogost. *Persuasive Games: The Expressive Power of Videogames.* The MIT Press, 2007.

[17] I. Bogost. The rhetoric of video games. *The Ecology of Games: Connecting Youth, Games, and Learning*, 2008.

[18] C. Brom, V. Sisler, and T. Holan. Story manager in 'europe 2045' uses petri nets. *Virtual Storytelling. Using Virtual Reality Technologies for Storytelling*, pages 38–50, 2007.

[19] M. A. Bruce and B. Borg. *Psychosocial frames of reference: core for occupation-based practice. Edition: 3.* SLACK Incorporated, 2002.

[20] F. Budinsky, M. Finnie, J. Vlissides, and P. Yu. Automatic code generation from design patterns. *IBM Systems Journal*, 35:151–17, 1996.

[21] B. C. and A. A. Petri-nets for game plot. *in Proc. of Artificial Intelligence and Simulation Behaviour Convention*, 3:6–13, 2006.

[22] P. Cox and T. Smedley. Experiences with visual programming languages for end-users and specific domains. In *OOPSLA Workshop on Domain-Specific Visual Languages*, pages 87–96, Tampa Bay FL, October 2001. Univ. of Jyvaskyla, Dept. of Computer Science TR-26.

[23] C. Crawford. Storytron: Interactive storytelling. `http://www.storytron.com/`.

[24] C. Crawford. Erasmatron. `http://www.erasmatazz.com/`, 1995.

[25] C. Crawford. Computer story generation system and method using network of re-usable substories. United States Patent 5604855, 1997.

[26] C. Crawford. *Interactive Storytelling.* New Rider Games, Berkeley, CA, 2005.

[27] M. Cutimisu, D. Szafron, J. Schaeffer, K. Waugh, C. Onuczko, J. Siegel, and A. Schumacher. A demonstration of scriptease interruptible and resumable behaviors for crpgs. In *National Conference on Artificial Intelligence (AAAI)*, pages 1968–1969, 2007.

[28] L. Daly. Natural Language Game Programming with Inform 7. O'Reilly ONLamp.com, LAMP: The Open Source Web Platform, `http://www.onlamp.com/pub/a/onlamp/2006/06/08/inside-inform-7.html?page=1`, 2006.

[29] G. Delmas, R. Champagnat, and M. Augeraud. Plot monitoring for interactive narrative games. In *ACE '07: Proceedings of the international conference on Advances in computer entertainment technology*, pages 17–20, New York, NY, USA, 2007. ACM.

[30] S. DeRose. Xml linking language (xlink). *Domain-specific languages: an annotated bibliography (2000), ACM SIGPLAN Notices archive*, 35, 2000.

[31] C. Dormann. Interview with Prof. Henry Jenkins. HOT Topics, Human-Oriented Technology Lab, Carleton University, `http://hot.carleton.ca/hot-topics/articles/jenkins-interview/`, 2006.

[32] C. Dormann, J. Fiset, S. Caquard, B. Woods, A. Hadziomerovic, E. Whitworth, A. Hayes, and R. Biddle. Repurposing a computer role playing game for engaging learning. In *Ed-Media: World Conference on Educational Multimedia, Hypermedia, and Telecommunications*, pages 4430–4435, Montreal, Canada, 2005.

[33] Dungeon magazine. Issue 1, Volume 1, TSR Inc., 1986.

[34] Dungeon submission guidelines. Dungeon Staff, Paizo Press, `www.paizo.com/writersguidelines/dungeon_writer_guidelines.pdf`, 2002.

[35] Ensemble Studios. Age of Empires 3. Microsoft Game Studios, 2005.

[36] Firaxis Games. Sid Meyer's Civilization 4. Take-Two Interactive Software, 2005.

[37] R. Firth. Cloak of darkness: Inform. Roger Firth's Inform Pages, `http://www.firthworks.com/roger/cloak/inform/index.html`.

[38] G. Florijn, M. Meijers, and P. van Winsen. Tool support for object-oriented patterns. *Lecture Notes in Computer Science*, 1241:472–495, 1997.

[39] G. Frasca. September 12: A toy world. Powerful Robot Games, `http://www.newsgaming.com/games/index12.htm`.

[40] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Professional, 1995.

[41] A. Glassner. *Interactive Storytelling: Techniques for 21st Century Fiction.* AK Peters, Ltd., 2004.

[42] H. Goldstein. America's Army Invades Xbox 360: Red Storm brings the next installment of the Army's official videogame exclusively to 360. `http://xbox360.ign.com/articles/795/795776p1.html`, June 11, 2007.

[43] T. Green and A. Blackwell. Cognitive dimensions of information artefacts: a tutorial. `www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDtutorial.pdf`, 1998.

[44] T. Green and M. Petre. Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of Visual Languages & Computing*, 7(2):131–174, June 1996.

[45] T. R. Green. Cognitive dimensions of notations. In *Proceedings of the fifth conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and computers V*, pages 443–460, New York, NY, USA, 1989. Cambridge University Press.

[46] G. Gygax and D. Arneson. Dungeons and Dragons: Rules for Fantastic Medieval Wargames Campaigns Playable with Paper and Pencil and Miniature Figurines, 3-Volume Set. Tactical Studios Rules, 1974.

[47] T. Hollingsed and D. G. Novick. Usability inspection methods after 15 years of research and practice. In *SIGDOC '07: Proceedings of the 25th annual ACM international conference on Design of communication*, pages 249–255, New York, NY, USA, 2007. ACM.

[48] K. Holtzblatt, J. B. Wendell, and S. Wood. *Rapid Contextual Design: A How-to Guide to Key Techniques for User-Centered Design (Interactive Technologies).* Morgan Kaufmann, 2005.

[49] E. Hutchins. *Cognition in the Wild.* MIT Press, 1995.

[50] T. Igarashi, J. D. Mackinlay, B. wei Chang, and P. T. Zellweger. Fluid visualization of spreadsheet structures. In *In Proceedings of IEEE Symposium on Visual Languages*, pages 118–125. IEEE, 1998.

[51] I. S. Inc. Deus ex. Eidos Interactive, 2000.

[52] Infocom. Infocom company history. `http://www.infocom-if.org/company/company.html`, 2009.

[53] Inform Fiction. Six standard examples. `http://www.inform-fiction.org/examples/index.html`.

[54] H. Jenkins. From serious games to serious gaming (part one): Revolution. Confessions of an Aca-Fan: Official Weblog of Henry Jenkins `http://www.henryjenkins.org/2007/11/from_serious_games_to_serious_1.html`.

[55] M. Kauhanen and R. Biddle. Cognitive dimensions of a game scripting tool. In *Future Play '07: Proceedings of the 2007 conference on Future Play*, pages 97–104, New York, NY, USA, 2007. ACM.

[56] M. Kauhanen, C. Eaket, and R. Biddle. Patterns for story authoring tools. In *In Proceedings of 12th European Conference on Pattern Languages of Programs (EuroPLoP 2007)*. Hillside Europe, 2007.

[57] M. Kauhanen, M. Tran, and R. Biddle. Examining authoring tools for serious games. In *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, pages 6091–6097, Chesapeake, VA, 2007. AACE.

[58] M. Kuniavsky. *Observing the User Experience: A Practitioner's Guide to User Research.* Morgan Kaufmann, San Francisco, 2003.

[59] P. J. Lucas and F. Riccardi. Concurrent hierarchical state machine. `http://chsm.sourceforge.net/`, 2007.

[60] J. Lytle. Korea aims for world video games domination: Government backing games industry to sell korea to the west. Techradar.com, `http://www.techradar.com/news/gaming/korea-aims-for-world-video-games-domination-491328?src=rss&attr=news`, December 6, 2008.

[61] M. Mateas and A. Stern. *Façade: An Experiment in Building a Fully-Realized Interactive Drama.* Game Developers Conference, Game Design Track, March 2003.

[62] D. Meichenbaum. Cognitive behavior modification: The need for a fairer assessment. *Journal Cognitive Therapy and Research*, 3(2):127–132, 1979.

[63] Microsoft PressPass. Xbox 360 registers biggest black friday in its history: Xbox 360 consoles and games sell at record levels; outsells playstation 3 by three-to-one ratio. Microsoft Corporation, `http://www.microsoft.com/presspass/press/2008/dec08/12-01Xbox360BlackFridayPR.mspx`, December 1, 2008.

[64] D. Moar. BioWare Aurora Neverwinter Nights Toolset Module Construction Tutorial. BioWare Corporation, `http://nwn.bioware.com/builders/modtutorial.html`, 2006.

[65] R. Molich and J. Nielsen. Improving a human-computer dialogue. *Commun. ACM*, 33(3):338–348, March 1990.

[66] B. A. Nardi. Concepts of cognition and consciousness: four voices. *SIGDOC Asterisk J. Comput. Doc.*, 22(1):31–48, 1998.

[67] National Instruments Inc. Introduction to labview: A six-hour course. http://zone.ni.com/devzone/cda/tut/p/id/5241, 2003.

[68] National Instruments Inc. Labview. http://www.ni.com/labview/, 2009.

[69] G. Nelson. 27. dwarves! (lines 3045-3148). browsing advent.inf. inform-fiction.org, http://www.inform-fiction.org/examples/Advent/Advent_2_27.html.

[70] G. Nelson. Natural language, semantic analysis and interactive fiction. In *IF Theory*. St Anne's College, Oxford, 2006.

[71] T. Nelson. *Literary Machines*. Eastgate Systems, Watertown, MA, 1982.

[72] J. Nielsen. How to conduct a heuristic evaluation. useit.com: Jakob Nielsen's Website, http://useit.com/papers/heuristic/heuristicevaluation.html.

[73] J. Nielsen. Ten usability heuristics. useit.com: Jakob Nielsen's Website, http://www.useit.com/papers/heuristic/heuristiclist.html.

[74] J. Nielsen. Usability inspection methods. In *CHI '95: Conference companion on Human factors in computing systems*, pages 377–378, New York, NY, USA, 1995. ACM.

[75] J. Nielsen and R. L. Mack. *Usability Inspection Methods*. Wiley, 1994.

[76] Nintendo Co. Consolidated financial highlights: Consolidated results for the six months ended september 2007 and 2008. Nintendo Co., Ltd., http://www.nintendo.co.jp/ir/pdf/2008/081030e.pdf#page=11, 2008.

[77] Nintendo Co. Consolidated sales transition by region. Nintendo Co., Ltd., http://www.nintendo.co.jp/ir/library/historical_data/pdf/consolidated_sales_e0809.pdf, 2008.

[78] J. Noble, A. Taivalsaari, and I. Moore. *Prototype-Based Programming: Concepts, Languages and Applications*. Springer-Verlag, Singapore, 1999.

[79] Video gamer segmentation report. The NPD Group, Inc., http://www.npd.com/corpServlet?nextpage=entertainment-special-reports_s.html, 2008.

[80] C. Onuczko, M. Cutumisu, D. Szafron, J. Schaeffer, M. Mcnaughton, T. Roy, K. Waugh, M. Carbonaro, and J. Siegel. A pattern catalog for computer role playing games. In *In Proceedings of GameOn North America*, pages 33–38, 2005.

[81] E. Packard and P. Granger. *The Cave of Time*. Prentice-Hall, 1979.

[82] R. Panko. What we know about spreadsheet errors. *Journal of End-User Computing*, 10(2):15–21, 1998.

[83] R. Pratchett. Gamers in the UK: Digital Play, Digital Lifestyles. BBC, http://crystaltips.typepad.com/wonderland/files/bbc_uk_games_research_2005.pdf, 2005.

[84] Global entertainment and media outlook: 2008-2012 industry previews. PriceWaterHouseCoopers International Limited, http://www.pwc.com/extweb/industry.nsf/docid/8CF0A9E084894A5A85256CE8006E19ED?opendocument&vendor=#video, 2008.

[85] D. Princess. Storytron (review). http://www.tale-of-tales.com/DramaPrincess/wp/?p=38, 2006.

[86] Procedural Arts. Façade: a one-act interactive drama. http://www.interactivestory.net/, 2005.

[87] A. Rahien. Building domain specific languages in boo. MEAP Release, http://www.manning.com/rahien/, 2008.

[88] D. Reily. Extreme gamers spend an average of 45 hours per week playing video games: New study profiles seven gaming segments by key behavioral metrics, usage patterns and purchasing preferences. NPD Inc., http://www.npd.com/press/releases/press_080811.html, 2008.

[89] S. Ruiz. Darfur is Dying. Interactive Media Program, School of Media Cinematic Arts, University of Southern California. Winner of Darfur Digital Activist Contest, sponsored by mtvU and Reebok Human Rights Foundation http://www.darfurisdying.com/, 2008.

[90] K. Salen and E. Zimmerman. *Rules of Play : Game Design Fundamentals.* The MIT Press, 2003.

[91] B. Sawyer and P. Smith. Serious games taxonomy. Serious games initiative, http://www.seriousgames.org/presentations/serious-games-taxonomy-2008_web.pdf, 2008.

[92] J. Schaeffer and D. Szafron. ScriptEase — A Scripting Language for Computer Role-Playing Games. University of Alberta, http://www.cs.ualberta.ca/~script/, 2003.

[93] Serious games initiative. Woodrow Wilson Center for International Scholars in Washington, D.C, http://www.seriousgames.org/.

[94] Sony Computer Entertainment Inc. Unit sales of hardware (since april 2006): Playstation 3 worldwide hardware unit sales. http://www.scei.co.jp/corporate/data/bizdataps3_sale_e.html, 2008.

[95] D. Spinellis. Notable design patterns for domain-specific languages. *Journal of Systems and Softwar*, 56(1):91–99, 2001.

[96] K. E. Steiner and T. G. Moher. Graphic storywriter: an interactive environment for emergent storytelling. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 357–364, New York, NY, USA, 1992. ACM.

[97] A. Sutcliffe and N. Mehandjiev. Introduction. *Commun. ACM*, 47(9):31–32, 2004.

[98] N. Szilas. Interactive drama on computer: Beyond linear narrative. In *In Proc. AAAI Fall Symposium on Narrative Intelligence*, pages 150–156. AAAI Press, 1999.

[99] B. M. Testa. Much more than a game: 'america's army' provides an enterprise platform for army training. Defense Systems, `http://defensesystems.com/articles/2008/05/much-more-than-a-game.aspx`, 2008.

[100] M. Tran. Distributed Cognition in Game Development: Influence of Socio-Technical Environment on Collaboration and Communication. Master's thesis, Carleton University, Ottawa, Ontario, June 2008.

[101] J. Tweet, M. Cook, and S. Williams. *Player's Handbook: Core Rulebook I (Dungeons & Dragons d20 3.0 Fantasy Roleplaying).* Wizards of the Coast, 2000.

[102] J. Tweet, M. Cook, and S. Williams. *Player's Handbook, Version 3.5 (Dungeon & Dragons Roleplaying Game: Core Rules).* Wizards of the Coast, 2003.

[103] US Army. America's Army: The Official Army Game. US Army, `http://www.americasarmy.com/`, 2009.

[104] R. A. Virzi, J. L. Sokolov, and D. Karis. Usability problem identification using both low- and high-fidelity prototypes. In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 236–243, New York, NY, USA, 1996. ACM Press.

[105] W3C. Extensible Markup Language (XML). `http://www.w3.org/XML/`, 2005.

[106] J. Watamaniuk. Introduction to the aurora winter toolset. BioWare Corporation, `http://nwn.bioware.com/builders/toolsetintro.html`, 2006.

[107] C. Wild. Adrift: Adventure development & runner - interactive fiction toolkit. `http://www.adrift.org.uk/cgi/new/adrift.cgi`, 2006.

[108] R. Williams. Using cognitive ethnography to study instruction. In *ICLS '06: Proceedings of the 7th international conference on Learning sciences*, pages 838–844. International Society of the Learning Sciences, 2006.

[109] B. S. Woodcock. An analysis of mmog subscription growth. MMOGCHART.COM 23.0, `http://www.mmogchart.com`, April 9 2008.

[110] N. Yee. Motivations for Play in Online Games. *CyberPsychology & Behavior*, 9(6):772–775, 2006.