University of Alberta

Quest Patterns for Story-Based Video Games

by

Marcus Alexander Trenton

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Fall 2009

Edmonton, Alberta

Examining Committee

Duane Szafron, Computing Science

Jonathan Schaeffer, Computing Science

Mike Carbonaro, Faculty of Education

Abstract

As video game designers focus on immersive interactive stories, the number of game object interactions grows exponentially. Most games use manually-programmed scripts to control object interactions, although automated techniques for generating scripts from high-level specifications are being introduced. For example, ScriptEase provides designers with generative patterns that inject commonly-occurring interactions into games. ScriptEase patterns generate scripts for the game *Neverwinter Nights*. A kind of generative pattern, the quest pattern, generates scripting code controlling the plot in story-based games. I present my additions to the quest pattern architecture (meta quest points and abandonable subquests), a catalogue of quest patterns, and the results of two studies measuring their effectiveness. These studies show that quest patterns are easy-to-use, substantially reduce plot scripting errors, and their catalogue is highly-reusable between games. These studies demonstrate ScriptEase generative quest patterns are a desirable alternative to manual plot scripting in commercial, story-based games.

Table of Contents

List of Tables

List of Figures

Chapter 1 - Introduction

1.1 - The Massive Video Game Industry

Video games are pervasive in our society. They can be found on a large variety of technology platforms – including cell phones, personal computers, and game consoles. Sixty-five percent of American households play video games [19]. The average gamer is thirty-five years old and has thirteen years of gaming experience, dispelling the myth that video games are solely for children and teenagers.

With such a large market, it should come as no surprise that video games form a multibillion dollar industry. *Halo 3*, the best-selling video game of 2007, earned $170 million dollars (US) in its first day of sale [25]. That amount is more than the $155 million dollar (US) record opening weekend for the movie *The Dark Knight* [9] or the record opening day for the novel *Harry Potter and the Deathly Hollows*, which sold 8.3 million copies earning an estimated $150 million dollars (US) [19] [60]. The popularity of video games has increased rapidly; the video game industry has tripled in size since 1996 [19]. With ten billion dollars in sales for 2004, the video game industry is larger than the Hollywood movie industry [59].

Video games can generate more revenue than their movie counterparts. In 1995, *GoldenEye*, a successful movie in the James Bond franchise, earned $300

million dollars (US) [33]. In 1997, a video game based on the movie, *GoldenEye 007* [22], was released. It sold over eight million copies [40]. The price of a video game varies with the retailer but fifty dollars (US) is a good, if conservative, estimate [31]. Therefore, the estimated revenue for the *GoldenEye 007* game is over $400 million dollars (US), exceeding the revenue of its movie counterpart by $100 million.

Similarities between video games and movies go deeper. The eight-digit budgets of modern video games allow high production values. Musicians now compose songs specifically for games, including the *World of Warcraft* [58] soundtrack. Famous actors provide voice acting for a game's characters, such as Samuel L. Jackson in *Grand Theft Auto: San Andreas* [23]. A game's release is preceded by several months of expensive multi-media advertising, as was done for *Empire: Total War* [18] in 2009.

The core content of games has also grown more sophisticated. While *GoldenEye 007* was a revolutionary game in 1997, it is primitive by today's standards. Modern games, like *Elder Scrolls IV: Oblivion* [16], render nearly photo-realistic graphics. Physics concepts, such as momentum and buoyancy, are directly incorporated into *Half-Life 2* [24]. In *World of Warcraft* [58], thousands of players can play the same game simultaneously in real time over the Internet. Customers' expectations grow higher each year. Unfortunately, creating the content for these multigigabyte-sized games increasingly takes more time. Development times exceeding three years are not unheard of; *Empire: Total War*

took five years to develop [6]. The increased sophistication and high development values of video games increases development costs.

1.2 - Scripting in Video Games

Scripting is a major bottleneck for the flow of ideas from the authors' imaginations to the game's content. Scripts control the interactions of objects in a video game – such as characters, furniture, items, and invisible markers for locations. The degree of abstraction and control are similar to a movie script dictating character dialogue and stage directions. Game scripts control simple interactions, such as opening a door when a character pulls a lever or revealing a ghost when the hero enters a room. Scripts select the available dialogue lines a player character (PC) or non-player character (NPC) can speak, based on criteria such as past actions and character attributes. For example, a script is used to ensure that a conversation with the villain cannot occur until the hero possesses a specific gemstone. Scripts are also used to remember characteristics related to PC actions, such as a game score or a list of places visited.

Video games provide a complex environment and little processing time for an artificial intelligence (AI). Many techniques that are labelled as AI in computer games are not considered as AI techniques in academia. Scripting is regularly used as a substitute, in which case the actions of an NPC are specified like in a flowchart: the goal is implicit. The human designer of the flowchart reasons how to accomplish that goal. Therefore, scripts are also used in more complex

scenarios on both small and large scales. In *Elder Scrolls IV: Oblivion*, scripts are used to control life-like NPC behaviours – such as running a shop during the day, sleeping at home at night, and finding food to eat. To create challenging opponents and helpful allies, scripts dictate NPC combat tactics and strategies which control armies with hundreds of individual units in *Starcraft* [51], direct a single creature with dozens of actions in *Neverwinter Nights* [32], and build cities – while managing a variety of resources – in *Warcraft III: Reign of Chaos* [57].

Scripting is also used to control plot in story-based video games. Such games frequently boast strong, intricate plots – typically allowing the player to control a single character or small group of characters that interact with an expansive world through exploration, manipulation of the environment, conversation, and combat. Players of such games frequently unravel, in parallel, many interconnected threads of the plot by performing quests for numerous NPCs. A *quest* is a single mission for the player of the game which is given its own heading in the player's game journal. Each quest can have multiple solutions, and some solutions can hinder other quests. The choices made during quests can affect the course of the story, resulting in a different ending. This dissertation focuses on expressing the plots of three popular, story-based games – 1) *Neverwinter Nights* [32], 2) *Elder Scrolls IV: Oblivion* [16], and 3) *Star Wars: Knights of the Old Republic* [49] – in a clear and concise model.

The plot of a non-interactive narrative, like a book, can be expressed as a story arc, like that of *The Three Little Pigs* depicted in Figure 1.1. In contrast, the

plot of a story-based game can be considered a flowchart, a decision tree, or even a decision graph, in which each decision point corresponds to an event in the game, like the example in Figure 1.2. These events vary in abstractness. Concrete events include the death of a specific NPC or the acquisition of an item by the PC. More abstract events include the PC becoming the leader of an organization or becoming famous. Story authors naturally aggregate individual plot events into more cohesive, self-contained units to ease comprehension. Example aggregations include book chapters, play acts, and television episodes. Plot events in games are commonly aggregated into quests; the example in Figure 1.2 illustrates a quest to retrieve an ancient book from a ruined castle and return it to a wizard. Many modern games, like *Elder Scrolls IV: Oblivion*, use an open-world design: PCs are not restricted to a linear story line. Instead, they are free to roam about the world and interact with a wide variety of game objects in almost any order. The game's plot graph must therefore be comprehensive and flexible enough to anticipate any meaningful action or event in the game and respond accordingly. Continuing the previous example, if the PC destroys the book and fails that quest, the overall plot of the story should continue. The game developer's challenge lies in modeling the progression of events in a plot graph that is easy to understand and edit by the story author yet retains the algorithmic simplicity required by the programmer who implements it.

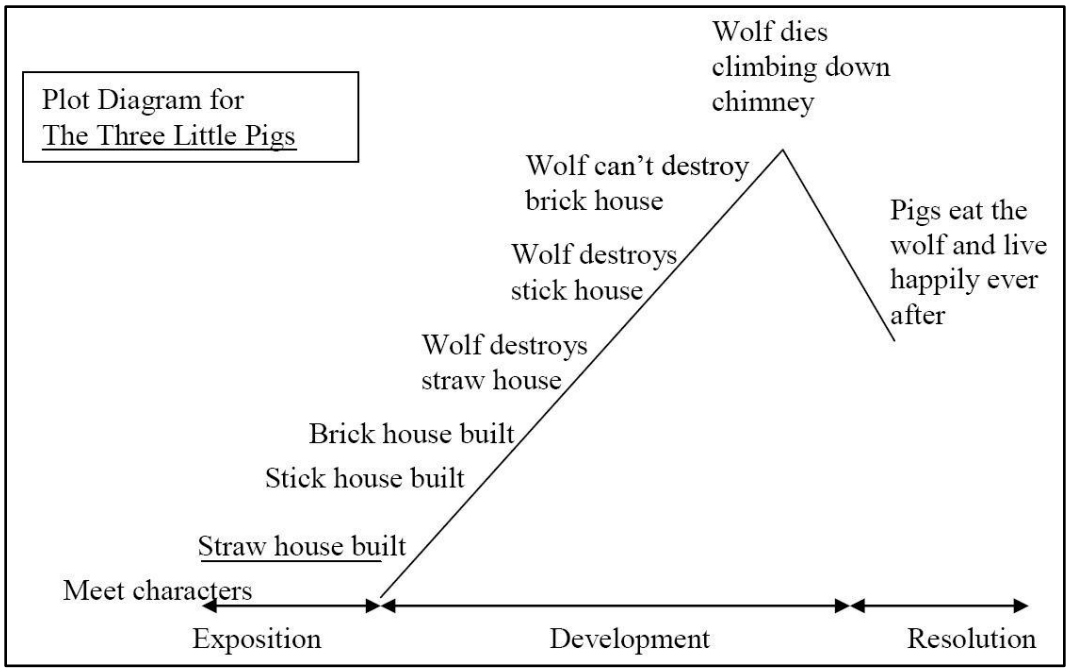


Figure 1.1 – The plot diagram for *The Three Little Pigs* [38]

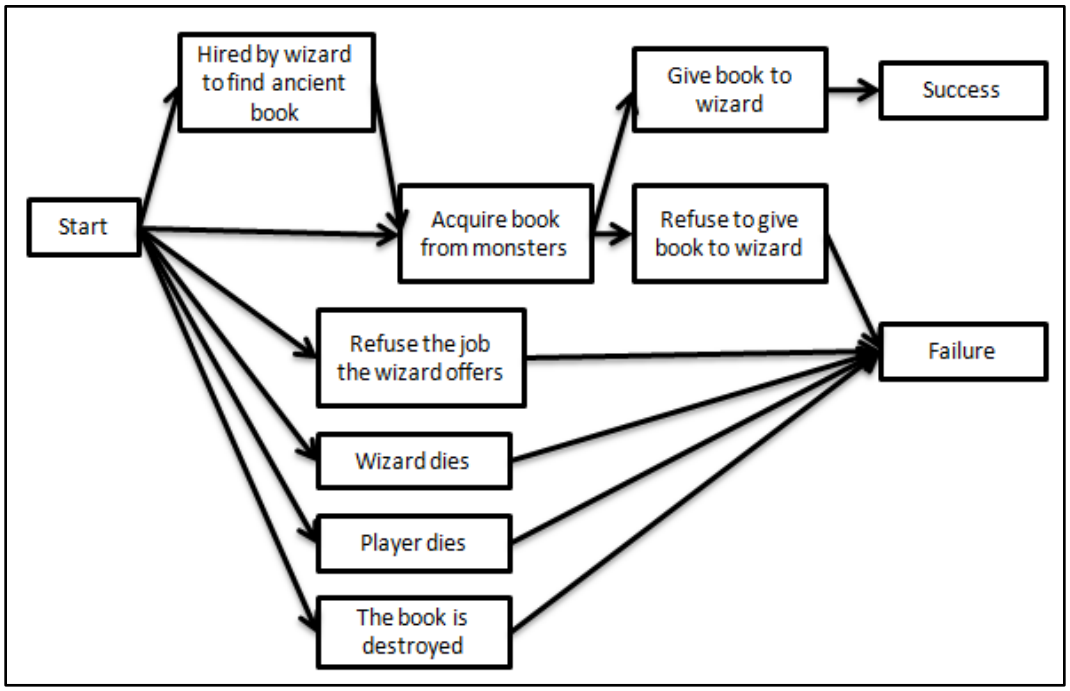


Figure 1.2 – A plot diagram for an interactive narrative

The plots of games are controlled through scripting. A scripting language provides an interface between the game's engine, which knows when a lever is pulled, and the author of a game story, who knows why the lever should be pulled and what should happen when it is pulled. Since scripts are changed and tested frequently, they are usually *interpreted*: at the moment of need during runtime each line of a script is translated into binary code and executed, slowing runtime performance. Alternately, scripts could be *compiled*: all scripts are translated in binary code before execution at runtime, slowing development. Some games use a general scripting language. For example, *Vampire: The Masquerade – Bloodlines* [56] and *Sid Meier's Civilization IV* [46] use the Python scripting language, while *Homeworld 2* [26] and *World of Warcraft* [58] use Lua. Other games use custom scripting languages. *Neverwinter Nights* [32], for example, uses NWScript, while *Elder Scrolls IV: Oblivion* [16] uses TES Script. Typically, these scripting languages are similar to the C or Java programming languages but possess much less functionality. The scripting code is usually written manually using a text editor, as opposed to using the more powerful tools available for programming languages.

Originally, video games were developed by small teams consisting of members with the ability to program and script. As the complexity of game production has increased, specialization of skills has occurred in the industry and most game story authors today have little or no computer programming (or scripting) experience. Either an author dictates story details to a programmer, who

writes scripts, or a technical designer is used as an intermediary. In either case, vague requirements and miscommunication between authors and script programmers can produce errors and delays.

Even after a game's release, scripts are still created and edited. Many games, such as *Neverwinter Nights* [32] and *Warcraft III: Reign of Chaos* [57], support user-created content to extend the lifespan of games and generate greater user interest. *Defense of the Ancients* [15] is an example of user-made playable content for *Warcraft III: Reign of Chaos* that is popular enough to be played in tournaments world-wide. To create this content, a user must write scripts to add meaningful interactions, but the complexity of scripting foils many users.

1.3 - Expanding the Use of Video Games

Video games have uses beyond entertainment. Flight simulators are used to teach flying. The United States Army developed the free, publicly-available video game *America's Army* as a recruitment tool [2]. Video games have also seen use in education. While some commercial video games (like *Sid Meier's Civilization IV* [46]) have limited educational value, some video games exist for education purposes.

Video games have some advantages over traditional teaching. The typing game *The Typing of the Dead* [54], adapted from the arcade game *The House of the Dead 2*, uses gameplay elements from commercial video games to make it more engaging than traditional typing software. Furthermore, a study in a Chilean

school showed that video games can improve the teaching of reading, spelling, and mathematics to first and second grade students [41]. The students readily learned how to use the technology, were more motivated to learn, and could progress at their own pace; the slowest student no longer delayed the entire class.

The development educational software is also bottleneck by the based complexity of scripting. One way to solve this problem is to reduce the difficulty of programming, allowing more game authors to become programmers. Programming environments like Alice [1] and Scratch [43] are iconic programming systems designed to simplify programming so even children are capable of writing programs. Instead of manipulating text directly, users manipulate icons and buttons that represent text. Such systems could also be used for creating simple games, but so far it has not been demonstrated that iconic programming languages can be used to author complex games.

1.4 - Thesis

Rather than lowering the complexity of programming, an alternate approach is to eliminate programming altogether. ScriptEase [29] (http://www.cs.ualberta.ca/~script/) realizes this alternate approach through automated script generation. Experimentation has already demonstrated that non-programmers can use ScriptEase to author simple stories [7]. While ScriptEase provides the scripting code, quest patterns provide ScriptEase with a model to represent the plots of story-based games.

I propose seven crucial qualities for a model (and its implementing tool) to successfully script plots in story-based video games that are based on the works of Cutumisu [11] and Spronck [48]:

1. **Adaptability** – The model should limit the author's imagination as little as possible. The model must be able to describe a wide variety of plots, including those developed for commercial, story-based games. If it cannot describe a specific plot element, there should be a mechanism to add that new plot element.
2. **Clarity** – The intent of any plot element and the resulting choices facing the player should be easily discerned from the model. Both the story's author and plot tool's implementer must understand the specifications of the plot element.
3. **Ease-of-use** – An author with no programming experience should be capable of using the model.
4. **Effectiveness** – Empirical evidence must show that the model is more effective than the alternatives.
5. **Reusability** - To reduce development time, the components of the model should be reusable within a plot element and within other plot elements.
6. **Robustness** – Since the narrative is interactive, the model should be able to accommodate whatever actions the characters perform, including those that change the intended order of events and introduce unexpected events.

7. **Scalability** – Describing an individual plot element is not enough. The model must be able to describe the relationship between elements clearly and effectively. Therefore, the entire narrative of a game should be easily represented by the model, regardless of the number of plot elements or the complexity of their relationships.

Unlike other plot models and tools used in academia and industry, I believe quest patterns in ScriptEase possess all of these qualities.

I have improved upon Curtis Onuczko's original quest pattern model [35], developed in 2007, by adding a new abstraction – *meta quest points* – and increasing the interactions between nested quests through *abandonable subquests*. These contributions enhance the reusability and scalability of quest patterns. These improved quest patterns were then used in a pair of studies to answer two questions: 1) whether quest patterns are representative of plots in commercial, story-based video games and 2) if quest patterns and ScriptEase are more effective than manual scripting. I contend that quest patterns implemented in ScriptEase are more reliable than manually-written scripts and can efficiently represent the quests found in commercial, story-based videogames. Therefore, quest patterns and ScriptEase can feasibly relieve the bottleneck of manually scripting content that afflicts the massive video game industry.

Chapter 2 - Related Work

While research has occurred in modeling dramatic structure [20] or common character patterns in Russian folk tales [39], little has been directly applied to interactive narrative, such as that found in video games. Academia, the video game industry, and even third-party users have investigated the issue. Some of the research focuses on using computer programs as a partial substitute for a human author. Another branch of research produces tools and models to assist an author. My research extends the latter branch, but it is instructive to discuss the former. The difficulties in using computers as authors led into the development of tools to assist authors.

2.1 - Computers as Authors

A player faces a large number of choices in a story-based game, resulting in a huge directional graph representing all the paths a plot can potentially take. In an effort to alleviate the author's burden of writing and programming for every possible contingency, computer programs can be used to implement the details of the author's plot outline. In addition, a computer program can easily add a degree of randomness so that the events of a plot will not unfold in a stale, predictable manner. There are a variety of methods for accomplishing that automation with varying degrees of author input.

Marc Cavazza *et al*. [8] use simulation to create an *emergent narrative*. Every NPC character possesses an artificial intelligence (AI) that attempts to

accomplish a goal. The NPCs simulate a dating scenario inspired by the *Friends* television sitcom using the Unreal Engine found in video games such as *Unreal Tournament* [55]. A plot emerges from the interactions of characters who act to fulfill goals instead of following an explicit plot. The drawback of this approach is diminished ease-of-use (since an author must program the AI for each of the NPCs) and of clarity (since the consequences of the player's choices may not be obvious from examining the NPCs' goals). Also, unlike typical video games, the player is not at the centre of the game, though the player can influence the outcome. In fact, if the player does nothing a narrative will still emerge.

If the player is to play a more central role in the story, another mechanism is needed to automate the details of an interactive narrative, such as a *drama manager*. A drama manager program manipulates the characters and sequence of events in a plot to maximize the anticipated enjoyment of the player. Several researchers and the commercial game *Left 4 Dead* [27] use drama managers to control the tension of the game.

Left 4 Dead uses a drama manager called a director. The game revolves around four players attempting to escape from a zombie apocalypse. The director dynamically places enemies, medicine, and ammunition in the levels. To provide rising and falling tension, the director places additional enemies and less aid if the players are doing well and vice versa. The game's developers termed this concept *procedural narrative*. The disadvantage of this approach is that it leaves little room for plot. The role of a writer is reduced to outlining the journey (such as, the

hospital can only be reached by travelling through the sewers) and writing single lines of random or contextual dialogue. Thus, procedural narrative lacks the adaptability to represent more complex plots.

To provide a stronger emphasis on plot, Mateas and Stern use the concept of *beats* to control the events in their interactive drama *Façade* [30], in which the player visits a couple with a deteriorating marriage. They describe a beat as "the smallest unit of dramatic action that moves a story forward" [30]. One of the beats in *Façade* is a discussion over the decoration of a room, during which the NPCs try to manipulate the PC into agreeing with them. Which NPC the player supports affects the outcome of the drama. Beats have a partial order and preconditions for their activations. When activated, a beat affects the story through the actions of the characters. *Façade* contains hundreds of beats, which the author must program. A beat manager, a type of drama manager, determines the next beat to be played partially based on the player's interactions, resulting in rising tension. *Façade* accomplishes the goal of making every run through the game feel unique by only using a few dozen of the hundreds of beats in each run.

However, Mateas and Stern admit that their system will not scale to larger applications. *Façade* took three person-years of effort to create a twenty-minute interactive drama. The immense amount of programming effort needed for a short game highlights a lack of ease-of-use. Another approach is needed for drama managers to be applicable to commercial video games.

Therefore Mateas, as part of the work of Sullivan *et al.*, assisted in the development of another drama management technique, *declarative optimization-based drama management* (*DODM*) [52], which uses the commercial video game *Neverwinter Nights* as a testing platform. Instead of choosing the next plot point (the counterpart of a beat) to play from a list, DODM manipulates the game world so the plot points occur dynamically. A good plot is well-paced, the player receives foreshadowing of future events, and all events are motivated by the player's knowledge of the game world. Sullivan *et al.* define a *goodness* score that the DODM attempts to maximize. For example, if it has been a long time since the last plot revelation, when the PC defeats an enemy, the DODM will cause that enemy to drop a note hinting at an item that is needed to escape the dungeon. The outline of the quest is still written by the author, while the foreshadowing and pacing of the plot is controlled by the DODM. This limitation of an author's control harms adaptability of the tool.

While the previous drama managers were concerned with pacing and tension, David Thue *et al.'s PaSSAGE* (*Player-Specific Stories via Automatically Generated Events*) adapts the story to the player's preferences through *delayed authoring* [53]. Delayed authoring infers a player's playing style (fighter, power gamer, tactician, method actor, or storyteller) as the game progresses and adapts the story's events to suit that style. For example, the event of discovering a murderer's identity could lead to an immediate reward if the player is accumulating wealth (a power gamer), a vigilante mission if the player is violent

(a fighter), or a puzzle to trap the murderer if the player previously chose to solve puzzles (a tactician). Using *Neverwinter Nights* as a platform, his study with one hundred and one participants has shown that those individuals less-experienced in gaming preferred the adaptive story, supporting the practical application of this line of research [53]. This is the only tool thus far that demonstrates its effectiveness with an empirical study. Considering that story-based games usually offer a variety of methods to complete a quest and PaSSAGE just infers the player's preferred method, the author still exerts great control over the plot, enhancing adaptability, unlike the previously described approaches. However, delayed authoring does not assist the author in writing all these contingencies. The tool still requires programming knowledge, and thus lacks ease-of-use.

Except for PaSSAGE, all of the approaches mentioned diminish the role of a human author in the video game creation process by reducing the author's control over the context of events. This leads to poor adaptability, which is one of the seven crucial qualities of a model and tool for plot. If an author no longer knows which NPC possesses a crucial item, when and where characters encounter each other, or how an argument escalates then the writing must forego such contextual details or must cover all possibilities. The former tactic leads to a generic or coarse plot (as seen in *Left 4 Dead)* while the latter overburdens the author (as seen in *Façade*), violating another crucial quality of ease-of-use. Authors have a critical role in story-based games, so replacing a human author is not feasible. Though PaSSAGE automates the modeling of the player's actions, it

manipulates the plot deterministically with at most five branches based on playing style. The author knows exactly how the plot is affected by PaSSAGE and can use the tool to assist scripting the plot. PaSSAGE illustrates the potential for tools that assist an author.

2.2 - Tools to Assist Authors

For commercial games, I believe better stories will arise from tools that assist an author rather than tools that replace an author. It is important for an author to decide which plot events happen and their effects on the game world. What is needed is a model for representing a plot that gives a human author full control while allowing a computer program to assist. Researchers and developers alike have suggested a variety of models and implemented several tools. Many of these tools apply to the commercial video game *Neverwinter Nights*, though the first two models described in this section do not. All of these tools illustrate the evolution of automated scripting culminating in the quest patterns I have refined and expanded upon.

2.2.1 - Petri Nets

Brom and Abonyi use a type of directed graph called a Petri net to model a quest [5]. Petri nets have a substantial body of supporting research and are used to describe complex processes in which multiple events can occur concurrently – an accurate description of a quest. Petri nets have an additional benefit of supporting the mathematical verification of outcomes, ensuring the robustness of a quest.

However, the complexity of Petri nets violates another crucial quality: ease-of-use. The roots of the graph are motivations of the PC and facts about the world, from time of day to object locations. Every meaningful action or event for every character is a node in the graph, resulting in complex graphs, such as the one shown in Figure 2.1 taken from Brom and Abonyi [5]. In the quest shown in the figure, the PC must obtain a puppet from a troupe of performers through theft, purchase, or friendship. The complication is that a member of the troupe is mistaken for a bandit. That simple plot is detailed with every possible complication that can arise during its execution, from the PC failing to summon the town's guards to the theatre burning down.

There is no level of abstraction between the quest as a whole and minute actions like "villagers going home." This one graph displays too much information: plot, behaviours, events, actions, and motivations. This lack of an intermediate abstraction makes this model too complex for an average author to use, which Brom and Abonyi concede. This lack of abstraction prevents the reuse plot elements within the story, as every plot element emphasizes context rather than common intent. Additionally, the goals of the quest and the choices the player can make are difficult to discern, greatly diminishing the clarity of model. The model could be refined through the experience gained from implementing this model for a game, but that has not happened. This approach is currently too complex to use. The following models use an abstraction between the level of quests and actions to simultaneously solve the problems of reusability and clarity.

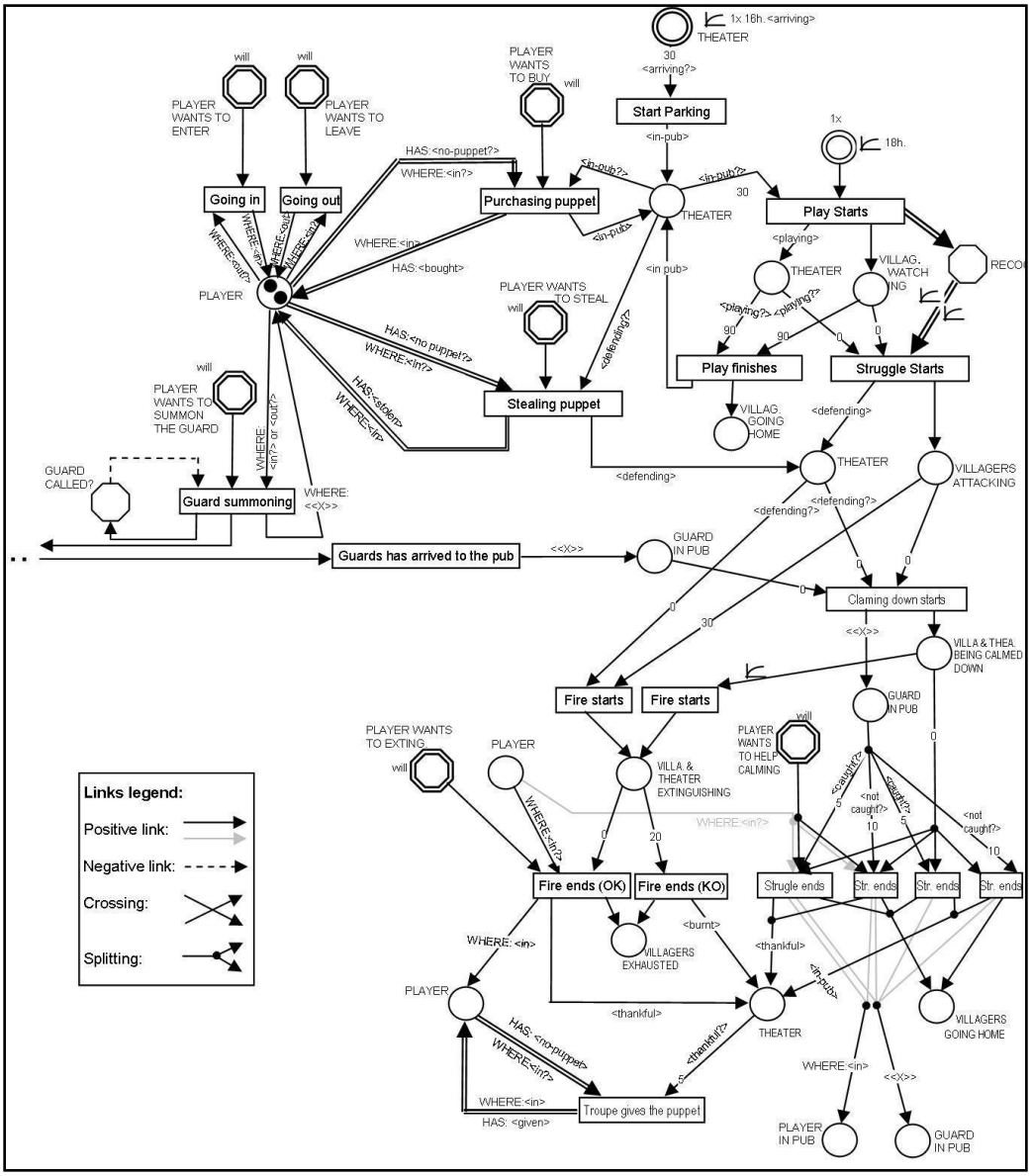


Figure 2.1 – Petri net model of a quest to obtain a puppet [5]

2.2.2 - Quests in Botfighters2

A simpler quest model is presented by Saar [42]. A quest is composed of quest events, like reaching a dialogue choice or the death of an NPC. A quest begins at the *start* quest event and finishes at the *end* quest event. Each quest

event contains conditions for its execution and the actions that occur on execution. These quest events are the abstraction that Brom and Abonyi needed. Each quest event can have multiple resolutions, each leading to a different subsequent quest event. Progress through the quest is restricted to a single path: the next quest event to fire must directly follow the one previously fired. Figure 2.2 shows the model behind the *Defeat Enemy for Money* quest [41]. This model has the benefits of simplicity, ease-of-implementation, and allows loops of quest events, but it still prevents the elegant modeling of quests in which a multitude of tasks can be accomplished in any order, which is a limitation in adaptability. In addition, this model does not allow connections between quests, harming scalability.

The testing platform is a simplified version of the massively multiplayer online game *Botfighters 2* from It's Alive Mobile Games. The program, which is integrated with the game's existing tools, includes a graphical user interface (GUI), shown in Figure 2.2, for an author to create quests without writing scripts. The GUI allows the author to create new quests, add or remove events in the quest, change the relationship of events, and specify the event's parameters. However, that is the limit of the GUI's power. It cannot change the conditions and actions contained within a quest event. A programmer is needed to change those conditions and actions, as well as add new types of quest events, which reduces ease-of-use of the model.

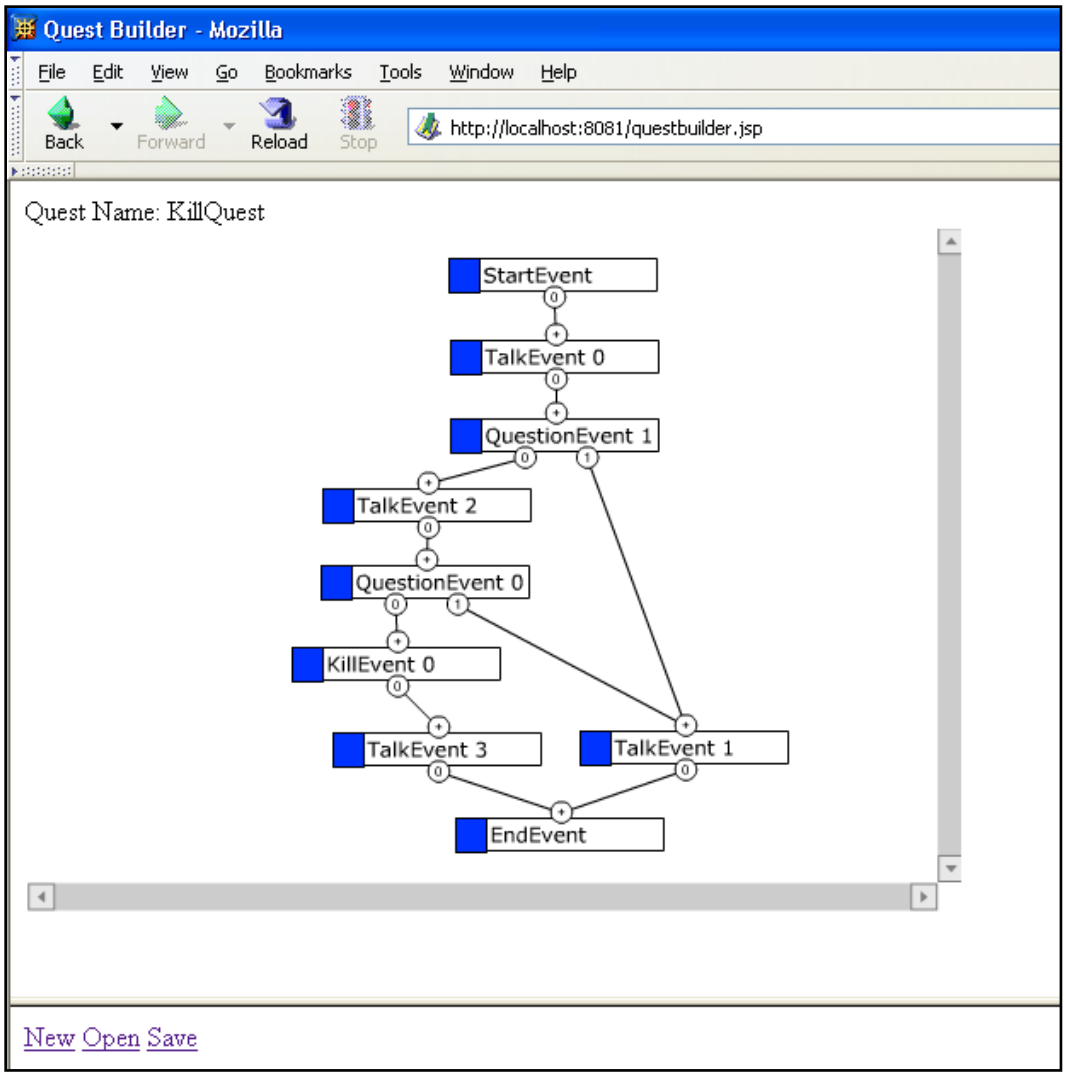


Figure 2.2 – The quest events and their relationships in the *Defeat Enemy for Money* quest [42]

While this work is a step in the right direction, it is underdeveloped. The author can only choose from a single type of quest and five types of quest events, so more quests and quest events are needed. The interface gives no feedback on invalid author choices. However, the idea of reusing quests and quest events among multiple instances is a good one, and ScriptEase practises that idea as well.

2.2.1 - Neverwinter Nights

Brom and Abonyi did not apply Petri nets to a commercial video game, and Saar used a simplified version of *Botfighters 2*. An important test of a model's effectiveness is whether it can function in a complete, commercially-available video game. Some of the computer-as-authors tools discussed previously use complete commercial video games as testing platforms. The only video game mentioned repeatedly is *Neverwinter Nights* (*NWN*), which is used in DODM and PaSSAGE. The next three tools to be discussed, Plot Wizard, Lilac Soul's *NWN* Script Generator [28], and ScriptEase [35], use *NWN* as well. Why is this game so popular for research? Video game developers usually refuse to provide the tools used to develop their games in order to protect trade secrets. One of the exceptions is Bioware Corp. [4], which not only released the Aurora Toolset used to develop *Neverwinter Nights* but even designed this toolset to be usable by the players of the game, affording researchers – as well as players – a platform for experimentation.

Since the discussion of the next three tools centres on their ability to construct quests for *NWN*, it is worthwhile to describe that game. *NWN* (Figure 2.3) is Bioware Corp.'s popular story-based game, winning 86 awards [32] since its release in 2002. The popularity of *NWN* was enhanced by providing the Aurora Toolset (Figure 2.4) with the game, which enables players to create their own stories (which are called modules in *NWN*). Thousands of user-made stories are available on-line. The top ten user-created modules have been downloaded more

than one hundred thousand times each. The Aurora Toolset supports visual tools for creating conversations, NPCs, environments, and game objects. What is lacking in the toolset is a simple way of designing interactions between game objects. Instead, a user must manually write scripts in the NWScript language using a text editor and a predefined list of API functions, variables, and constants (as shown in Figure 2.5). The NWScript language resembles C but with fewer features; there are no debugging tools and script errors fail silently.

Figure 2.3 – *Neverwinter Nights* [32]

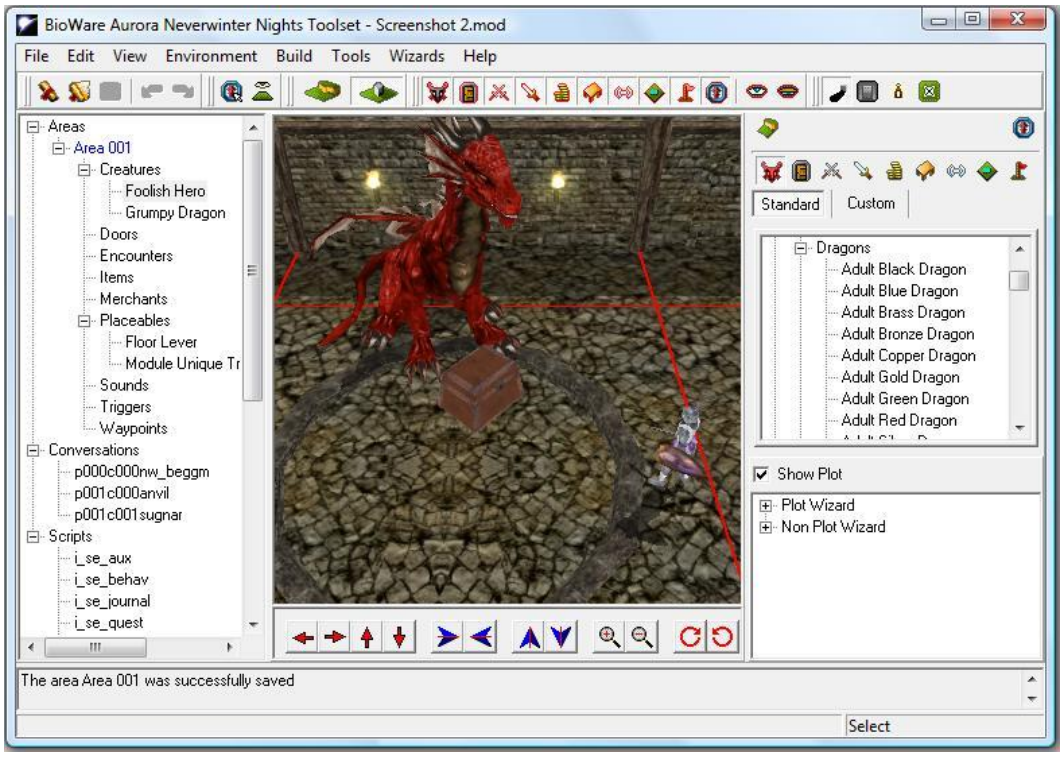


Figure 2.4 – The Aurora Toolset for *Neverwinter Nights*

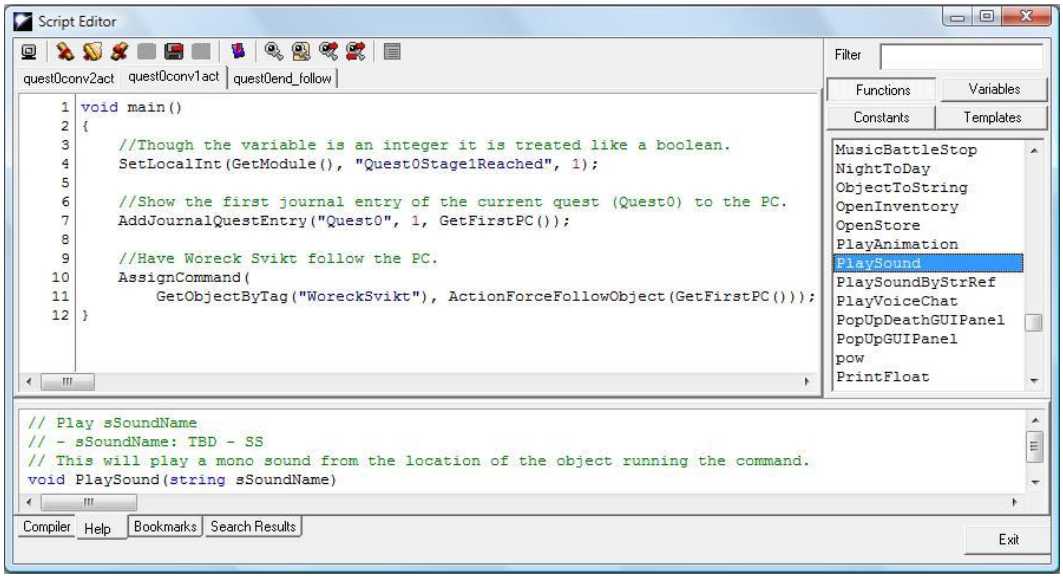


Figure 2.5 – NWScript in the Aurora Toolset

2.2.2 - Plot Wizard

The Aurora Toolset contains a tool for scripting quests: the Plot Wizard shown in Figure 2.6. The assumption behind its design is that the quests would be developed before the rest of the module. It provides four quest templates and the ability to create more customized quests. The author specifies the plot giver, the villain(s), extra(s), and prop(s). These objects are chosen from a catalogue of every object possible, rather than those already present in the module. After the wizard has finished, the toolset can easily refer to and place these objects.

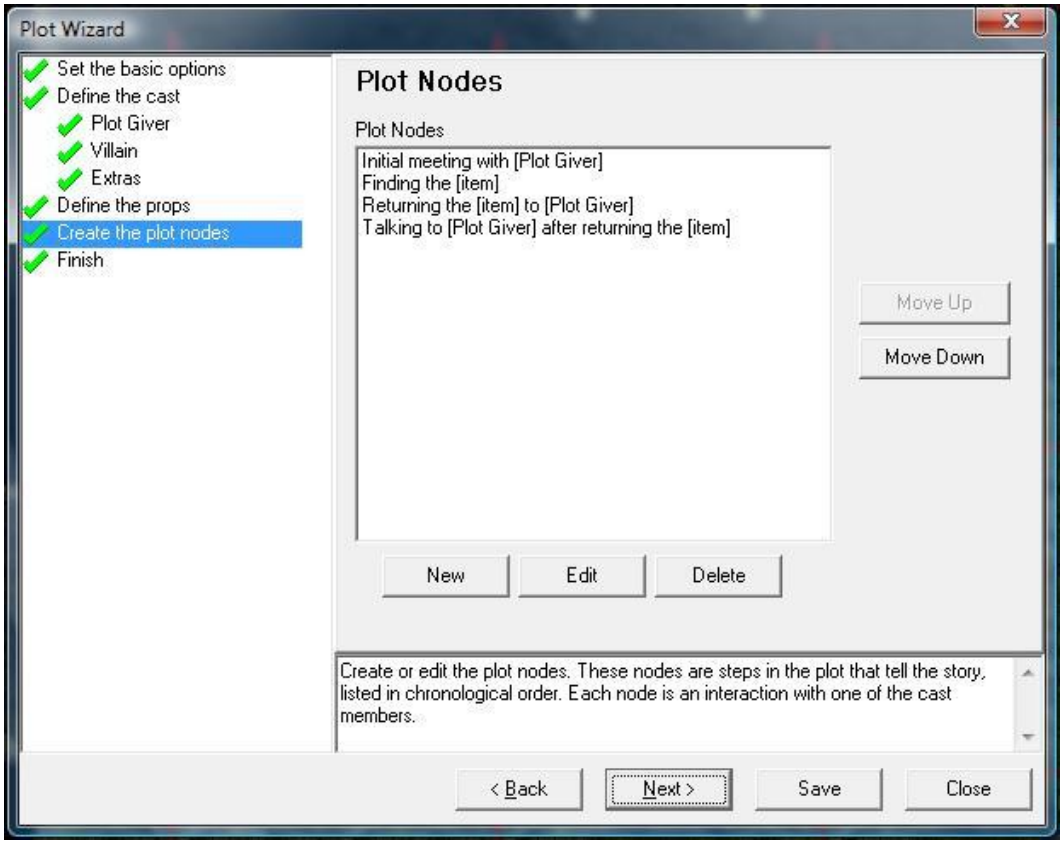


Figure 2.6 – The Plot Wizard in the Aurora Toolset with the plot nodes of a quest revealed

Despite its integration with the toolset and the meaningful defaults in the templates, the wizard has some significant weaknesses. The main weakness is the small catalogue of templates and their lack of adaptability. The four templates provided are *Assassination*, *Assassination (no proof required)*, *Fetch and Deliver*, and *Fetch and Retrieve*. This small list is actually even smaller than it appears due to a lack of reusability. The two assassination templates are just variations of each other; the wizard is too inflexible to allow for such variations within the same template. Similarly, the only difference between the two fetch templates is whether the receiver of the item is the same NPC that asked the PC to fetch the item in the first place. There is a mechanism to add new templates to this limited catalogue, but there are other limitations to the wizard.

Each quest is composed of plot nodes, the counterparts of Saar's quest events. The four plot nodes that compose a *Fetch and Retrieve* quest are shown in Figure 2.6. Each plot node represents an interaction between the PC and a cast member of the quest. There are some safeguards to prevent logical errors: the villain (who presumably dies during the quest) cannot be used in the final plot node as the NPC that gives the PC a reward for the completing the quest. The types of interactions are fixed at talking with an NPC, killing an NPC, acquiring an item, and unlocking a placeable (such as a chest or door). This fixed list insufficiently represents all the potential plot nodes present in the story included with *NWN*, such as entering an area or using a lever. Each plot node has a rigid structure, preventing the addition of other actions. For example, while it is

possible to use the plot wizard to script that the acquisition of a magical sword from a stone causes a journal entry to appear, it is impossible to add a lightning visual effect, which is a limitation in adaptability. Even the ordering of plot points is inflexible and fallible; the intent is that each plot node is completed sequentially from top to bottom, but it is possible to skip some plot nodes. If a player acquires the item used by the *Fetch and Retrieve* quest before being told by an NPC to retrieve that item, then the PC will skip that conversation plot node. Instead, the PC receives a journal entry advising them to return the item to its original owner, even though the PC has not met the owner and does not know to whom the item belongs. Thus, the model shows a lack of robustness.

Another significant limitation of the wizard is that each quest must be linear, a lack of adaptability. There is no way to specify in the wizard that there are two possible methods of completing a quest (two branches). This stifles an author significantly, since tension and choice are two essential components of drama. For example, the choice of resolving a quest peacefully or violently is common in story-based games, including *NWN*. While it is possible to work around this limitation by manually editing scripts produced by the wizard, such changes would require programming knowledge that fewer and fewer story authors possess. In addition, such changes will be overwritten if the wizard is subsequently used to edit that quest. Though useful for a niche of commonly-used quests, the Plot Wizard tool is not adaptable enough for general use.

2.2.3 - Lilac Soul's *NWN* Script Generator

While the Plot Wizard is useful for quests, any scripting tool can potentially be used to manage plot. Lilac Soul's *NWN* Script Generator (abbreviated as Lilac Soul Generator) [28] is a general-purpose, publicly-released, third-party automated scripting program. Like the Plot Wizard, the Lilac Soul Generator uses plain-English terminology as much as possible instead of scripting terminology, as shown in Figure 2.7. This tool has been downloaded over one hundred and thirty thousand times, which demonstrates the demand for a script generator that does not require programming knowledge.

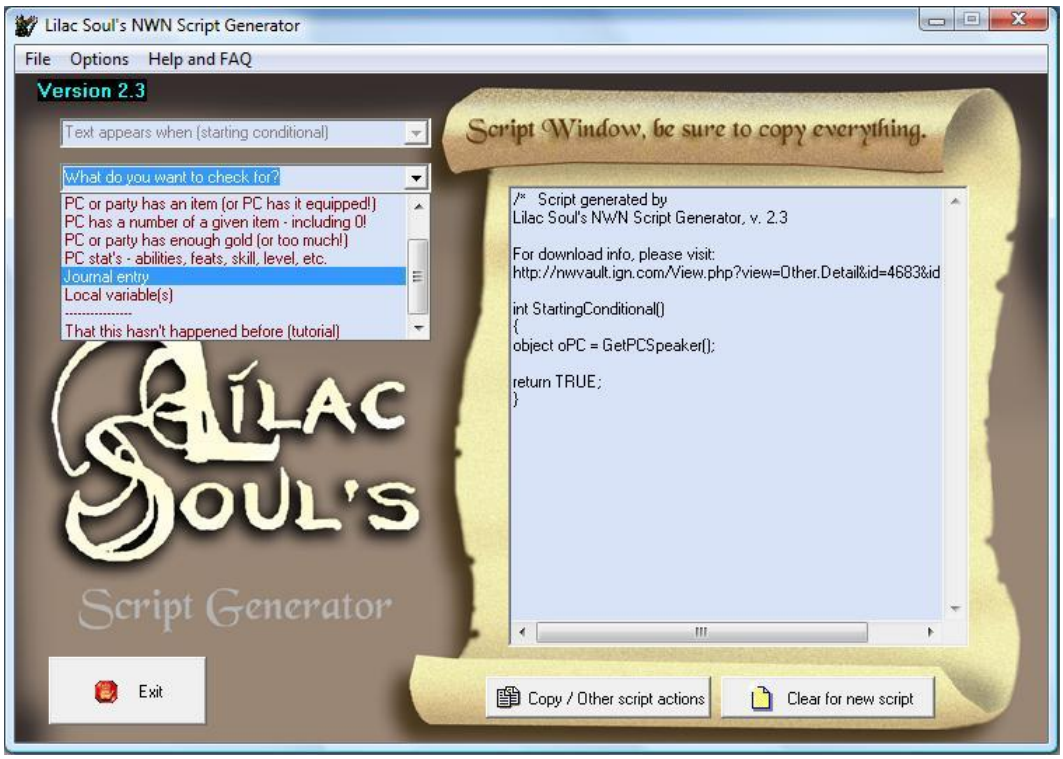


Figure 2.7 – Lilac Soul's *NWN* Script Generator

Although the catalogue of script templates is much larger than the Plot Wizard's, it still lacks the power of manual scripting. The catalogue only contains events caused by the PC or by an NPC's heartbeat (a timer that fires every six seconds while the creature is alive), preventing the tool from scripting plots concerning interactions between NPCs. Unlike the Plot Wizard, the catalogue is fixed so further additions are impossible. The adaptability of the tool suffers as a result.

Also, unlike the Plot Wizard, the Lilac Soul Generator is not completely integrated with the Aurora Toolset. Manual user action and knowledge are required to correctly install the generated scripts using the Aurora Toolset. Scripting knowledge is still required in some cases, diminishing ease-of-use for non-programmers. For example, the script that fires when the PC acquires any item requires the user to manually merge script fragments dictating what happens when the PC acquires a specific item.

Though it is possible to use the Lilac Soul Generator to script quests, it is not easy. While the Plot Wizard incorporates the high-level concepts of a quest and plot points into its design, the Lilac Soul Generator does not. Instead, low-level concepts (such as events and actions) must be used. Each aspect of a quest, from conversation control to displaying journal entries, must be scripted independently; no mechanism exists to organize these scripts for ease of reference. The result is that all scripts controlling quests, behaviours, and

miscellaneous reactions are jumbled together, increasing the difficulty of editing generated scripts for particular quests at a later time. Thus, the tool lacks clarity.

2.2.4 - Previous Version of Quests in ScriptEase

Since the tools designed by the game developers and third-party developers were unsatisfactory in automatically scripting quests, in 2007 Curtis Onuczko applied a more formal approach using the ScriptEase tool [35]. He applied the concept of software design patterns [21] to quest design [36]. A software design pattern is a human-readable, reusable guide for solving a common programming problem. The guide is often written in pseudocode. Quest patterns are simple enough to be understood and created by a game author while also simple enough to be easily implemented by a programmer. Quest patterns are expressed in a hierarchy of patterns; quests patterns are composed of quest point patterns (the counterpart of the Plot Wizard's plot nodes and Saar's quest events) that are, in turn, composed of encounter patterns. Encounter patterns describe actions that occur in response to a specific event. For example, when the villain is killed, the encounter makes an appropriate entry in the PC's journal. Such an encounter pattern is used to construct a *Kill* quest point pattern as part of an *Assassinate* quest pattern. Though these patterns could be applied to generate any scripting language, the ScriptEase tool generates NWScript of *NWN* as a proof of concept.

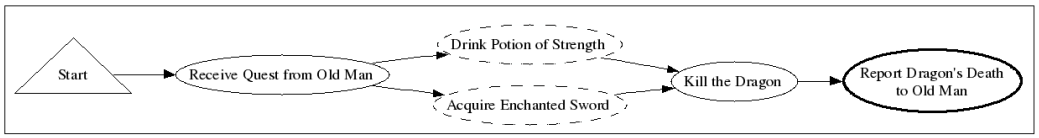


Figure 2.8 – Quest pattern diagram for a quest to defeat a dragon [35]

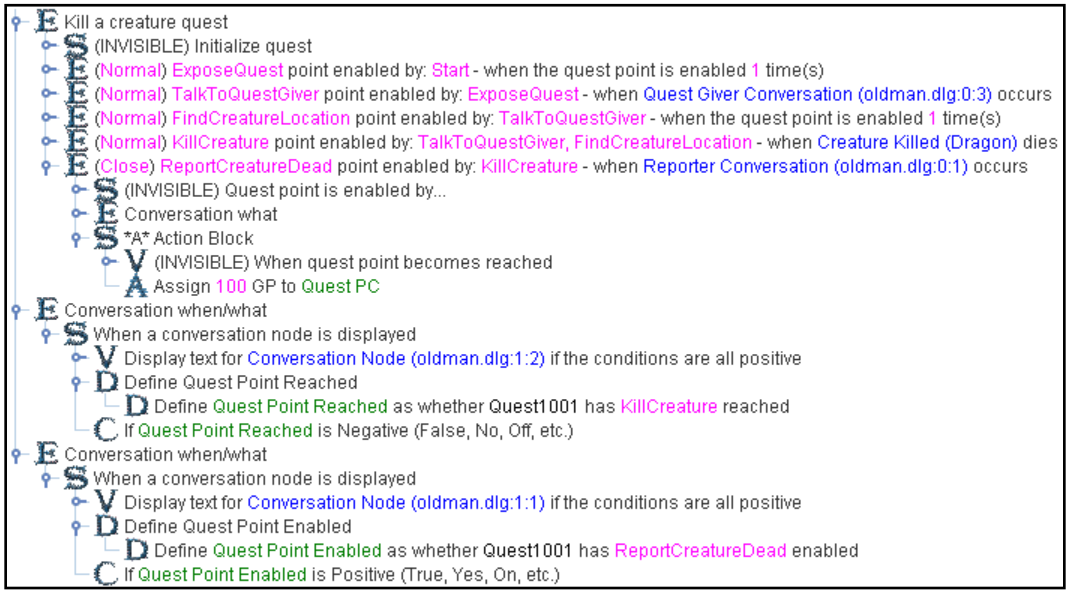


Figure 2.9 – ScriptEase's implementation of the defeat a dragon quest, from Onuczko [35]

ScriptEase's quest patterns improve over the quest templates of the Plot Wizard in many ways. ScriptEase is designed to be as powerful as manual scripting, in contrast to the restrictive Plot Wizard and Lilac Soul's Script Generator. In quest patterns, the relationships between quest points are represented in a directed graph of events, as opposed to the Plot Wizard's linear chain, so that quests in which the PC must accomplish several goals in any order are now expressible. The catalogue of patterns – quest, quest point, and encounter

– is easily extensible. The patterns are also adaptable so that each quest can be customized to what the author wants: the addition of new quest points to an existing quest or new visual effects to an encounter is easily accomplished. Unlike the Lilac Soul Generator, ScriptEase integrates with the *NWN* script compiler so that scripts created with ScriptEase can be immediately played in *NWN*, without any additional work in the Aurora Toolset. However, ScriptEase is not as integrated as the Plot Wizard: ScriptEase and the Aurora Toolset cannot work on the same module at the same time. Unlike the Plot Wizard or the Lilac Soul Generator, ScriptEase can detect when a quest point fails at runtime (for example a *Converse* quest point fails if the NPC to be conversed with dies), give an appropriate journal entry, and determine if the quest as a whole can still be completed. Quest points can also be marked as optional, allowing some quest points to be skipped. Chapter 3 - ScriptEase and Quest Patterns explains how these advantages are accomplished.

Onuczko demonstrated the power of his quest patterns by expressing all of the quests found in the Beggar's Nest area of *Neverwinter Nights* (nine in total) as quest patterns. These quests required 93 adaptations (changes to the original pattern) to generate 255 lines of scripting code that would have otherwise been written manually. He argued the effort required in making an adaptation to a pattern in ScriptEase is less than that needed to write and test a script line, since automatically-generated scripts are free of the programming errors that afflict manually-written scripts.

Onuczko's quest patterns meet all seven of the crucial qualities for a plot model. However, there is still room for improvement in reusability, scalability, and effectiveness. The quest patterns included in Onuczko's dissertation [35] are unsophisticated, providing an opportunity for additional abstraction in quests and quest points. His quest pattern catalogue was admittedly incomplete, containing only five patterns: *retrieve/deliver an item quest*, *retrieve/deliver multiple items quest*, *retrieve/deliver one of multiple items quest*, *talk to quest*, and *kill a creature quest*. This limited catalogue provides opportunities for abstraction between patterns, since there are repetitions of similar *retrieve* quest patterns. To express all the quests found in *NWN*'s included story, let alone other commercial games, more quest patterns are needed. The effectiveness of these patterns was demonstrated for *NWN* but not for other commercial, story-based games. Though a hierarchy of quests and subquests can be defined, they only serve as aggregations of quest points, though more interaction between a superquest and subquest is possible.

2.3 - Contributions

Of all the research and tools reviewed, quest patterns in ScriptEase provide the most functionality. However, ScriptEase still suffers from the same problem as the other tools: a lack of reusability in patterns and templates. Seemingly minor variations – whether proof of an assassination is required or whether the receiver of an item is the same person who sent the PC to fetch it – require separate quests.

My contributions improve four of the seven crucial qualities of plot models list in Section 1.4 - Thesis. My research solves the reusability problem of multiple quest patterns being variations of each other. Furthermore, my research improves the scalability of quest patterns and demonstrates their adaptability and effectiveness through a pair of studies. Specifically, I make five contributions to quest patterns, whose details are explained in the next two chapters:

1. An abstraction of a quest point, the *meta quest point*, which enhances the reusability of quest patterns by allowing the quest designer to specify an intent for a quest point while letting the author fulfill that intent by matching the context of the story
2. A consistent, practical, and scalable mechanism for hierarchical quest construction by abandoning expired subquests
3. An adaptable and comprehensive quest catalogue representing a wide variety of quests found in commercial story-based video games
4. A case study that shows the effectiveness of the quest catalogue through high ratings of coverage, precision, utility, and usage across three of the most popular story-based games of recent years
5. A user study empirically validating that quests authored using quest patterns are more reliable than those manually scripted, further demonstrating the effectiveness of quest patterns

Chapter 3 - ScriptEase and Quest Patterns

In contrast to other graphical programming tools (like Alice), ScriptEase does not seek to teach programming to a video game story author that otherwise has no use for it. Instead, the idea behind ScriptEase is that a tool should use the language of the user instead of inventing a new language that the user must learn.

ScriptEase and quest patterns work together to develop a story from the imagination of the author to the finished work presented to the player. Quest patterns model quests for a video game at a level of abstraction that an author can understand and use. Quest patterns are also specific enough that a programmer can implement them. In the case of ScriptEase, they are specific enough that code can be automatically generated from them. Though the two ideas of modelling and scripting may appear distinct, they are closely related in this case. ScriptEase uses a hierarchy of patterns to model and implement its automated scripting. The quest patterns' model extends this hierarchy, but so do other patterns, like conversations and behaviours. Quests are just a specific type of pattern used to express plot. Therefore, greater insight into the workings of quest patterns can be gained by first examining how ScriptEase uses patterns in general and then examining quest patterns in greater detail, including my meta quest point and abandonable subquest creations.

3.1 - ScriptEase

ScriptEase allows an author to create story interactions in a top-down manner. The author starts with an abstract intent embodied by selecting a pattern [21] and creating an instance of it. The pattern provides high-level specifications on how to realize this intent. The author then adapts the pattern to the context of the game's story in a hierarchical manner, starting from general options and proceeding to specific details. ScriptEase uses this adapted pattern to generate the scripting code needed to accomplish this intent in the game. If the patterns in the catalogue are insufficient for the current story in the game, an author can construct a new pattern and add it to the pattern catalogue for reuse in the current story and future stories. Although the pattern catalogue is game independent, the current implementation of ScriptEase only generates NWScript code for *Neverwinter Nights* (*NWN).* However, this implementation is sufficient to show the utility of using generative patterns in story-based games.

ScriptEase's procedure is the opposite of bottom-up manual scripting, in which the author creates expressions that are placed into statements that are placed into scripts. In fact, with ScriptEase an author always selects from a small number of explicit options and never has to create any kind of expression or other construct on a blank page. With ScriptEase's top-down approach an author can select a high-level pattern without considering its details. A high-level pattern often provides default values for the details, which may be changed using menus.

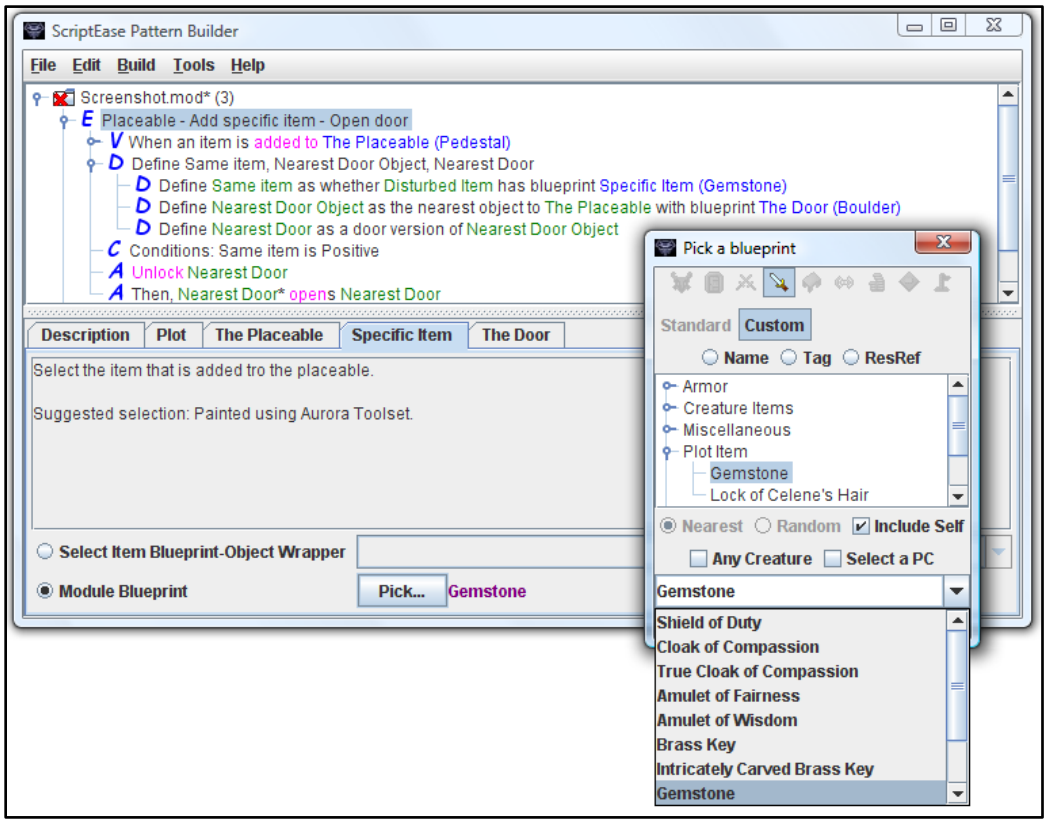


Figure 3.1 – An encounter pattern in ScriptEase opened to reveal its contents and a menu for the encounter's *Specific Item* option

For example, suppose an author wants to create an interaction that when a *Gemstone* is placed on a *Pedestal* a *Boulder* slides to reveal a cave opening. This is an example of an *encounter pattern* in which actions occur in response to an event. The author would begin by selecting the intent, the *Placeable – Add specific item – Open door* pattern from the encounter pattern catalogue [29] to create an instance of the pattern, as represented by the line in Figure 3.1 that starts with the script letter *E*. The encounter pattern in Figure 3.1 is open to reveal its contents, but the author does not need to open it to use it. The author only needs

to adapt the encounter pattern instance, which is done by selecting choices for the pattern's options: *The Placeable* used, the *Specific Item* added to *The Placeable*, and *The Door* opened. Figure 3.1 shows the author selecting the *Gemstone* as the *Specific Item* option from a menu. The author has already selected the *Boulder* as *The Door* option and the *Pedestal* as the *The Placeable* option. The plain English used in ScriptEase highlights the pattern's clarity.

Figure 3.1 illustrates that an encounter pattern can contain four types of basic components: *events* (*V*), *definitions* (*D*), *conditions* (*C*) and *actions* (*A*). These components are the bottom of the pattern hierarchy. An *event* is an occurrence in the world, like disturbing the inventory of the *Pedestal*. A *definition* is the equivalent to an assignment statement in a programming language and can use a variety of concepts:

- A dynamic game object, a state or object statistic that may change during runtime (such as the door closest to the *Pedestal* or the PC's wealth),
- The result of mathematical or logical operation, and
- Answers of true or false to questions (like whether the item added to the *Pedestal*'s inventory is the *Gemstone*).

An *action* also encompasses multiple concepts:

- One or more calls to game engine that affects objects (for example, opening a door as shown in Figure 3.1),
- Changes to the game's interface (such as starting a conversation), and

- Recording current information (like how many doors the PC has opened).

A *condition* provides additional constraints on whether the subsequent actions occur. The example in Figure 3.1 has a condition ensuring that only the *Gemstone*, and no other item, will trigger the actions when it is added to the *Pedestal*.

An author only needs to know about the contents of an encounter if the contents must be changed or when a new encounter pattern is being designed. For example, if the author wanted to adapt the encounter in Figure 3.1 further by firing a visual effect when the door is opened, the author could open this encounter and add a visual effect action after the action that opens the door. Adding an action involves selecting it from a menu and setting its options – in this case, the kind of visual effect and its target. These encounter patterns are easy-to-use. A case study [29] showed that high school students, many without programming experience, could use encounter patterns.

Encounter patterns are the building blocks for more sophisticated patterns, including behaviours, dialogue, and quests. Behaviour patterns are used to specify which tasks are performed by NPCs and when these tasks can be initiated, interrupted, resumed [14], and learned [13]. Behaviours can be used to create more realistic environments, such as a busy tavern [14] or foes that learn to fight better [13]. Dialogue patterns [47] can be used to control conversation by generating scripts that determine which lines can be spoken under which

conditions. However, this dissertation focuses on quest patterns that are described in the next section.

3.2 - Quest Patterns

Many story-based games have a nonlinear plot. The plot can be considered a large decision graph that is traversed as the story unfolds. It specifies and controls the potential decisions a player can make at each point in the story. The graph can be divided into units called quests. A *quest* is single mission that the PC can or must complete. Quest patterns build upon the hierarchy of patterns used in ScriptEase to implement these quests and their decision graphs. This hierarchy and the mechanisms used to develop successively more complex plots are described in this section.

Quest patterns focus on the journal entries found in story-based games. Usually in such games (like *Neverwinter Nights*, *Elder Scrolls IV: Oblivion*, and *Star Wars: Knights of the Old Republic*), an in-game journal lists active and completed quests. When a significant event occurs in the quest, the journal updates to summarize the quest's progress and remind the player what the PC should do next. These significant events are termed *quest points*, the counterpart of the Plot Wizard's plot events or Saar's quest events.

3.2.1 - Quest Points

A quest is composed of *quest points*. For example, a *Defeat the Dragon* quest can be divided into three quest points: *Conversing* with a villager to learn

that the dragon is a menace, *Killing* the dragon, and *Conversing* again with the villager to report success. This example is represented by the *Exterminate* quest pattern. A simple diagram of this pattern is shown in Figure 3.3, with the legend in Figure 3.2. A ScriptEase representation of the pattern is shown in Figure 3.4, in which a *Q* represents a quest, a ▪ (bullet) represents a quest point, and the *Converse* quest point is opened to reveal its encounter pattern components.

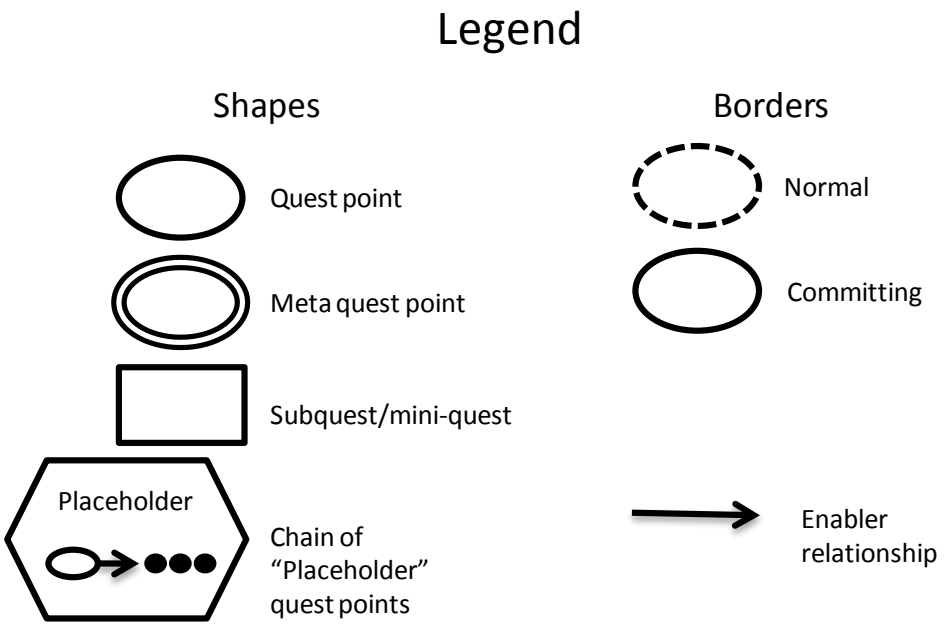


Figure 3.2 – Legend for quest pattern diagrams

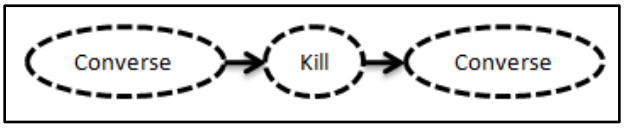


Figure 3.3 – An *Exterminate* quest pattern instance representing the *Defeat the Dragon* quest

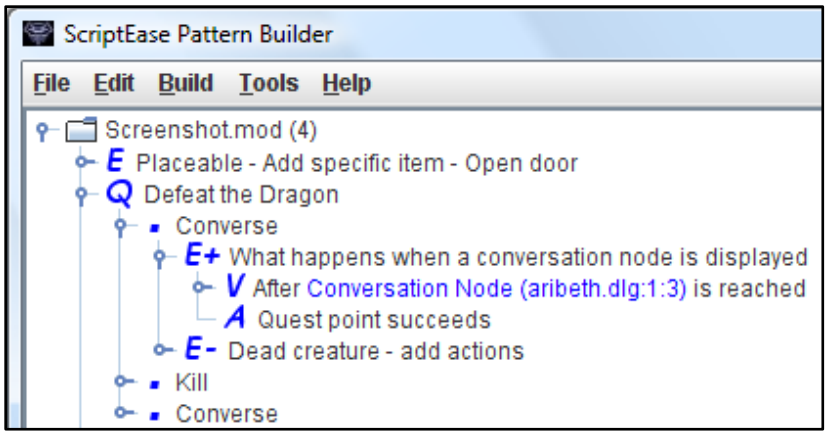


Figure 3.4 – An *Exterminate* quest pattern implemented in ScriptEase, with the *Converse* quest point opened to reveal its success and fail encounters

3.2.2 - Options

Options in ScriptEase are choices for configuring a pattern. Just as encounter patterns have options to set (such as the *specific item* in Figure 3.1), quest patterns and quest point patterns have options as well. Every quest pattern has two common options, a name and a failure journal entry that is displayed if the quest fails. Quest patterns can also have options unique to the individual pattern. The *Exterminate* quest pattern's unique options are the victims and how many of the victims must be killed. In the *Defeat the Dragon* example, the unique options are set so that the victim is a dragon and PC must kill only one. Each quest point pattern has many options that are described in this section:

- Name,
- Success and failure journal entries,

- Commitment (whether this quest point is mutually exclusive with other quest points of the quest),
- Experience points awarded (the power of a character is measured through experience points),
- Gold awarded,
- Immediately preceding quest points,
- How many of those preceding quest points must succeed before this quest point is enabled, and
- Unique options for each quest point.

For example, the unique options for a *Converse* quest point are the NPC to be conversed with and the dialogue line that marks the quest point's success.

Although there are many options for quest points, two mechanisms automate option setting: defaults and inheritance. For example, the number of victims option has a default of one in the *Exterminate* quest pattern. A pattern can inherit the choices for unique options of the patterns that contain it. For example, the *Kill* quest point inherits the victim and the number of victims automatically from the *Exterminate* quest's options. Similarly, an encounter pattern can inherit options from the quest point or quest it is contained in. Inheritance allows a choice made in a high-level pattern to automatically set options at multiple low-level patterns, saving the user effort and reducing the opportunity for error.

3.2.3 - States of a Quest Point

Quest points measure how far the player has progressed in a quest and the player's available paths for future progress. Thus, the state of quest points and the mechanisms for state transitions are the crux of this quest model. A quest pattern guides a player through a quest by *enabling* quest points at the appropriate points in a story. An *enabled* quest point can potentially be the next quest point *reached* by the player. When a quest point is reached it may enable its subsequent quest points. The first quest point of a quest is automatically enabled. Figure 3.5 shows the first few steps of the *Defeat the Dragon* quest if it progresses in the intended order. For example, step one in Figure 3.5 shows that the first *Converse* quest point is enabled at the start of the story. This means that the PC can converse with the quest-giving NPC at that time. An enabled quest point becomes *reached,* and ceases to be enabled, when it either *succeeds* or *fails*. Each quest point has at least one encounter that if *fired* causes a quest point to succeed (*E*+ in Figure 3.4), and zero or more encounter patterns that if fired cause a quest point to fail (*E*-). For example, if the PC reaches a specific line of dialogue with the quest giver (step two of Figure 3.5) the first *Converse* quest point succeeds (step three), and if the quest giver dies before the conversation takes place then the *Converse* fails. Success and failure have their own journal entries.

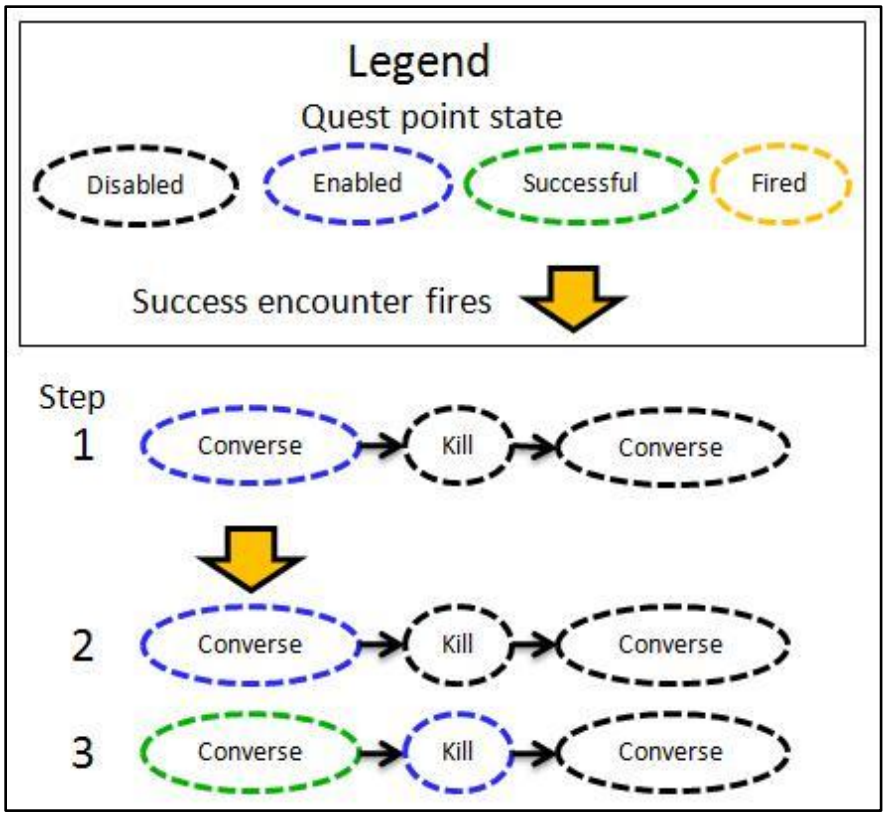


Figure 3.5 – The intended progression of the *Defeat the Dragon* quest

When a quest point is reached its successor quest points may become enabled. A quest point maintains a list of which quest points can enable it and how many of those quest points must be successfully reached (*succeed*) before it is enabled. By default, for linear quests in ScriptEase when a quest point succeeds, the subsequent quest point is enabled. As depicted in step three of Figure 3.5, when the first *Converse* quest point succeeds the *Kill* quest point becomes enabled. When *Kill* succeeds the second *Converse* – the *end* quest point – is enabled. When the *end* quest point succeeds, the quest succeeds. If a quest point fails then it enables no other quest points. If there is no longer a way for the

end quest point to succeed – such as when no quest points are enabled – then the entire quest *fails*. The condition for a quest's success is explicit, but the condition for failure is implicit and more complex to compute.

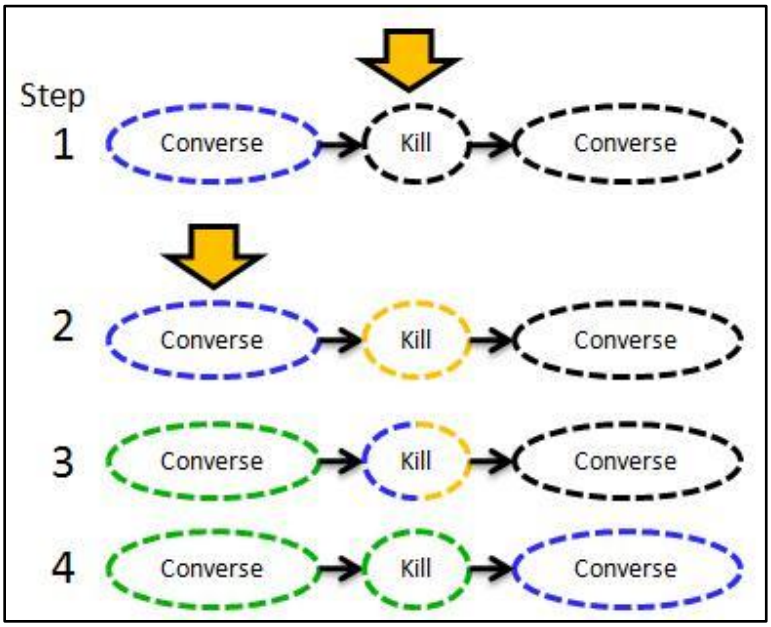


Figure 3.6 – An alternate progress for the *Defeat the Dragon* quest

A quest point cannot be reached if it is not enabled. What happens if a quest point's success encounter fires before the quest point is enabled? Figure 3.6 illustrates such a situation for the *Defeat the Dragon* quest. In that example, assume the PC kills the dragon before conversing with the villager (step one of Figure 3.6). In this case, the *Kill* quest point is not enabled when the dragon is killed. The generated scripts record that the *Kill*'s success encounter fired (step two) but do not mark the *Kill* as reached, since it was not enabled. For a quest point to succeed it must be enabled and one of its success encounters must fire.

However, immediately after the PC *Converses* with the quest giver to receive the quest (the yellow arrow in step two), the Converse succeeds (in step three) and the *Kill* quest point is enabled. Since its success encounter has already fired and is enabled (represented by both a blue and yellow outline in step three), the *Kill* quest point immediately succeeds (step four). Therefore, the PC can perform the encounters in the first two quest points of the *Exterminate* quest in the opposite order without breaking the quest, demonstrating the robustness of quest patterns.

The separation of the encounter success and quest point success allows quests to progress even if the success encounters occur in an alternate order. ScriptEase automatically generates the complex scripts that determine at runtime whether any quest point is enabled, fired, reached, successful, or failed. This automation aids the design of new patterns, whether by an author or by a designer of a new quest pattern. Their task is reduced to specifying the success and failure encounters for each quest point and the enabling relationships between quest points.

3.2.4 - Meta Quest Points

Many of the tools reviewed in the previous chapter have a problem with flexibility, causing poor reusability. Conceptually similar quests were represented with separate patterns or templates. The current iteration of quest patterns solves this problem through abstraction. For example, the *Exterminate* pattern instance of Figure 3.3 is very specific – it both starts and ends with a *Converse* quest point.

However, similar quests often start and end in a variety of different ways. For example, in the *Defeat the Dragon* quest the author may instead want the PC to arrive in a village to witness the menacing dragon leaving, and after the dragon is defeated to have a villager reward the PC with some jewellery. In this case, the author may want to start the quest with an *Arrive* quest point and end the quest with a *Converse – give items* quest point as shown in Figure 3.7, instead of starting and ending the quest with *Converse* quest points shown in Figure 3.3.

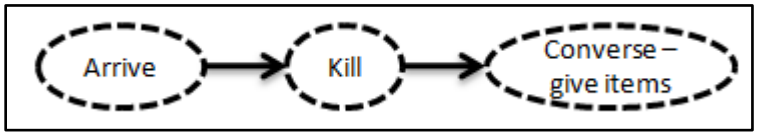


Figure 3.7 – Another *Exterminate* quest pattern instance

Although the quests in Figure 3.3 and Figure 3.7 have only one quest point in common (*Kill*), the quests are so conceptually similar that it is inefficient for them to be instances of two separate quest patterns. I created *meta quest points* as an abstraction to solve this reusability problem. A meta quest point can be used anywhere that a quest point can be used. A *meta quest point* is an abstract quest point that can easily be adapted to one of a small set of manually-selected quest point patterns or quest patterns, or to any other quest point, if necessary. Setting a meta quest point to a quest pattern is one way of forming a subquest, which are discussed in a later section. Meta quest points provide unlimited scope for structural change, since a meta quest point can be adapted to an empty node, a single quest point, or a subquest with variable length and structure.

Every quest starts and ends, which is where we see the most variation in quests. Therefore, I created a *Start* meta quest point whose intended choices include *Converse* and *Arrive*, and an *End* meta quest point that includes the intentions *Converse* and *Converse – give items*. Figure 3.8 shows the *Exterminate* pattern using meta quest points, along with two possible instances. Meta quest points have a double-dashed border both before and after being adapted to a specific quest point. Another example of a meta quest point occurs in the *Search* quest pattern, where the *Discover* meta quest point includes both *Converse* and *Approach* for greater adaptability.

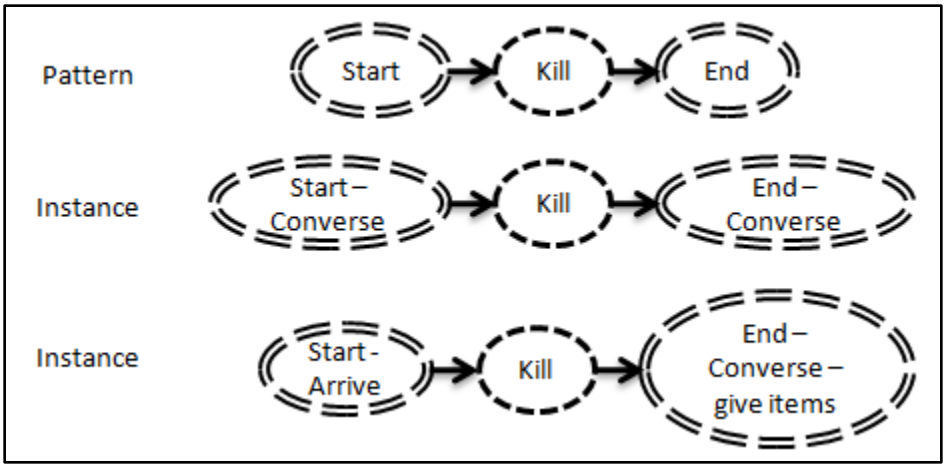


Figure 3.8 – *Exterminate* pattern using meta quest points with a pair of possible instances

An unadapted quest point succeeds as soon as it is enabled, thus functioning as a meaningless placeholder. For example, if the author wants no reward and no recognition after the PC kills the dragon the *End* meta quest point

would remain unadapted. The *End* meta quest point would succeed (thus completing the quest) as soon as the preceding *Kill* quest point succeeds. Figure 3.9 shows ScriptEase's implementation of this quest, in which ¤ represents a meta quest point. The *Start* quest point is adapted but the *End* is not.

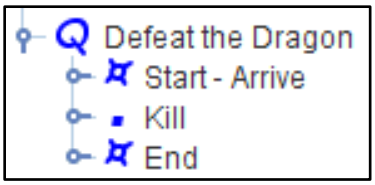


Figure 3.9 – An *Exterminate* quest instance in ScriptEase that contains one adapted and one unadapted meta quest point

3.2.5 - Branching

Our example *Exterminate* quest pattern is linear, but many quests in interactive story-based games are non-linear. There are two reasons for branching structures in a quest. The first is using dead-end branches to represent optional events. The second is providing more than one method to accomplish a goal. Quest patterns can support both kinds of branching.

An example of a dead-end branch is when a PC acquires a sword to help defeat the dragon. Though the sword is not required to defeat the dragon, the sword can still be an important part of the plot if the villager tells the player of its existence and utility. A dead-end quest point is not referenced by any other quest point's enabler list, so when it succeeds no other quest points are enabled. Figure

3.10 demonstrates a dead-end branch that uses an *Acquire* quest point. When the *Converse* succeeds it enables both the *Acquire* and *Kill* quest points.

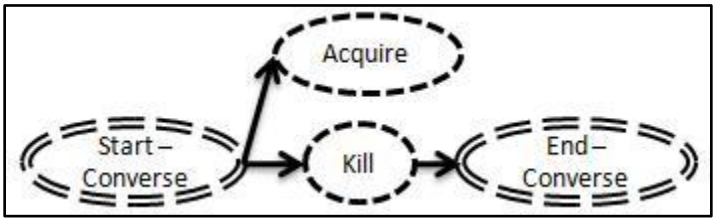


Figure 3.10 – An Exterminate quest with an *Acquire* quest point as a dead-end branch

Branches do not need to be dead-ends. Suppose the dragon problem could also be resolved peacefully by persuading the dragon to leave the village. Figure 3.11 shows this more general quest that splits into two branches and then merges back together: one branch contains a *Kill* quest point and the other contains a *Converse – use skill* (e.g. *intimidate*, *diplomacy*, *medicine*, etc…) quest point. The notation of "1 enable" on the final *Converse* quest point conveys the author's intent that only one of the two preceding quest points to succeed to enable it.

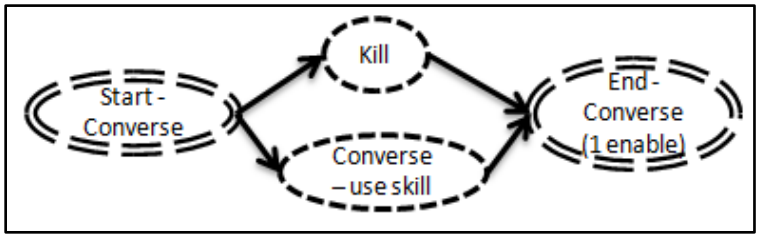


Figure 3.11 – A non-linear quest with two branches

For the example in Figure 3.11, the same conversation occurs regardless of whether the dragon was removed peacefully or violently. This was done deliberately to keep to example simple. If a conversation tailored to each branch is desired, a *Converse* can be added to the end of each branch and the *End* can be left unadapted.

The additional choices for completing the quest complicate the logic for determining if the quest fails or succeeds. Although the PC's lack of skill may cause the *Converse – use skill* quest point to fail, the *Kill* quest point must also fail for the quest as a whole to fail. ScriptEase generates scripts to support this logic without programmer intervention, by enabling the second *Converse* quest point when either quest point succeeds or by failing the quest if both quest points fail.

3.2.6 - Subquests

The plot of a game consists of many quests and they are not all independent, leading to the concept of *subquests*. A *subquest* is a quest pattern that is used as a single quest point in another quest, including as a choice for a meta quest point. Subquests allow the plot of a story to be represented in a scalable manner by a hierarchy of subquests. When the subquest is enabled as a quest point in the superquest, the first quest point contained in the subquest is enabled. When a subquest succeeds/fails it succeeds/fails as a quest point in the superquest.

For example, suppose the dragon cannot be persuaded to leave by a simple conversation. Instead, the dragon will only leave voluntarily if it receives a gemstone the villagers stole from it. The acquisition and delivery of the gemstone can be represented by a *Retrieve* quest pattern. The evolved dragon quest can now be represented by the *Do one of many* quest pattern instance shown in Figure 3.12, in which the player can choose between the *Retrieve* and *Exterminate* subquests. A subquest is represented as a rectangle. A *Join* quest point represents the point in the story when the PC has completed the needed number of branches. The *Join* quest point succeeds as soon as the required number of enabling quest points succeed, one enable in the case of the quest in Figure 3.12. The purpose of a *Join* quest point is to give a journal entry when numerous independent tasks have been completed and the player may not remember what to do next. In this case of the dragon quest, the *Join* will succeed as soon as either the *Exterminate* or *Retrieve* subquest succeeds and will write a journal entry reminding the player to report back to the quest giver. The ScriptEase view of this pattern is shown in Figure 3.13 with the subquests expanded. Since the ScriptEase view is a tree view, it does not explicitly display the branching and merging of the quest. However, viewing the enablers of the *Join* quest point shows that quest is not linear and the “Minimum # Enablers” field shows that it only needs one preceding quest point to succeed before it is enabled.

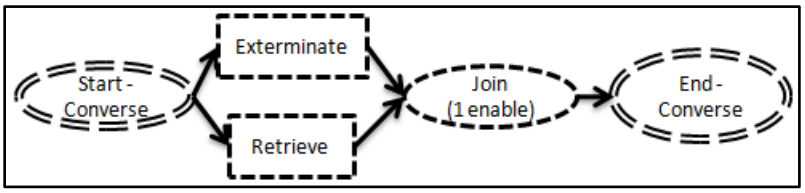


Figure 3.12 – *Do one of many* quest pattern with subquests

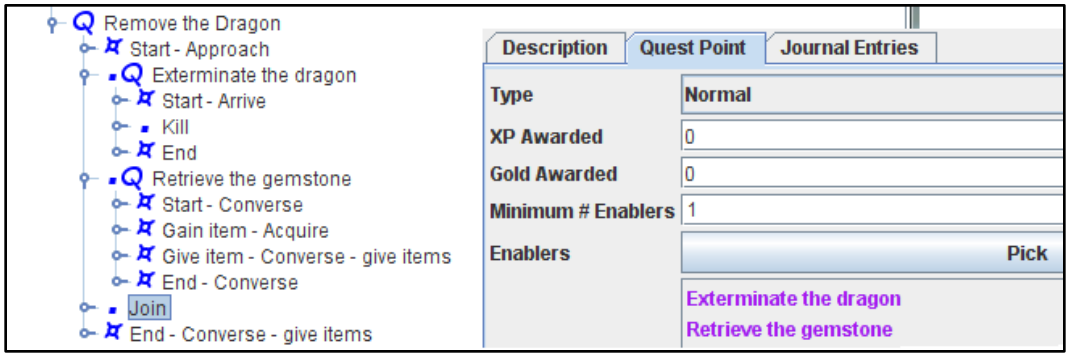


Figure 3.13 – Subquests and non-linear quests in ScriptEase

A quest point can only succeed or fail, but a subquest has an additional outcome: *abandonment*. In the dragon quest example, if the player has completed the quest through the *Exterminate* subquest then the dragon is dead and there is no point in completing the *Retrieve* subquest. Therefore, no further progress in the *Retrieve* subquest should occur. I created the concept of abandoned subquests to support this notion. If a subquest is marked as *abandonable*, when the superquest completes all quest points in the subquest are disabled and an abandonment journal entry is written. Why not just fail that subquest? Separating the journal entries based on failure or abandonment allows those journal entries to contain greater context, increasing adaptability. Using a failure journal entry would falsely imply that the quest ended because the PC made an error, instead of

completing an alternate solution. Why not just leave the quest unresolved? That solution is also unsatisfactory. Abandonable subquests prevent the journal from filling with subquests that can no longer advance the plot. The player is no longer falsely told by these journal entries that they should complete those redundant subquests.

A variation of a *subquest* is a *miniquest*. The journal entries for each subquest appear under different headings from the superquest and each other. All of the journal entries for a miniquest appear under the same heading as the superquest. Otherwise, a miniquest operates identically to a subquest. The distinction gives the author greater control over the number of quest headings in the journal and hints to the player how conceptually separate the quests are. Thus, a single quest (from a player's perspective) can be constructed from multiple quest patterns (from an author's perspective).

3.2.7 - *Quest completed* Quest Points

Quests can be related to each other in ways that are not easily expressed in a hierarchy of subquests. For example, let us return to the dragon quest of Figure 3.11: the PC can resolve the dragon problem through violence or diplomacy. What if the villagers will not entrust this dangerous quest to any wandering stranger but only to the person who rescued several of their children from bandits (the *Captive Children* quest)? What if the dragon will not listen to the PC unless it knows the PC is already a skilled negotiator, like the one that resolved a merchant

feud? Modelling the *Captive Children* quest as a subquest of *Remove the Dragon* would work, although the hierarchy would get more complex if this subquest is used in multiple other quests. For example, the *Captive Children* quest could be a prerequisite to another quest, *Orphaned Children*. It may be impractical to make a single quest a subquest for two distinct quests. Another quest prerequisite mechanism is needed. In addition, subquests are insufficient, since they cannot guard the middle of one quest based on the completion of another. For example, the author wants the *Merchants' Feud* quest to guard the diplomatic conversation with the dragon, but not to stop the player from receiving the quest. An awkward solution would be to directly express the enable relationship between quest points belonging to different quests. This solution will become more difficult and tedious as the number of quests in a game grows, so a more elegant solution is needed.

The *Quest completed* quest point solves this problem by allowing a quest point to be guarded by the completion of any quest. This enhances the scalability of quest patterns, since there no longer needs to be a hierarchical connection between a prerequisite quest and the quest point it guards. A *Quest completed*'s success encounter fires when the specified quest succeeds. This method is simpler than the usual enabler relationship, since picking a quest from a list of all quests is much easier than picking a quest point from a list of all quest points in all quests. Figure 3.14 shows how *Quest completed* quest points can be used. A *Quest completed* quest point for the *Captive children* quest and another for the

Merchants' feud quest are used to specify the relationship between those quests and the *Remove the Dragon* quest.

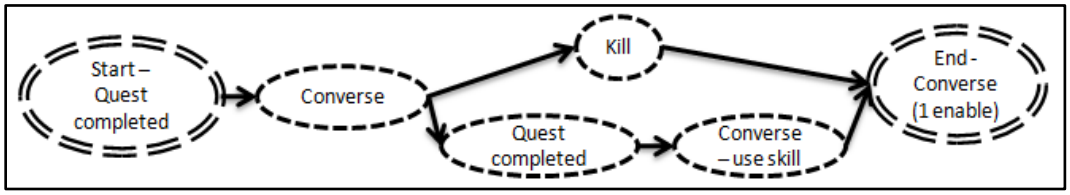


Figure 3.14 – The *Remove the Dragon* quest using *Quest completed* quest points to provide a relationship with other quests without using subquests

However, there is a potential problem with this solution. What is stopping the PC from conversing with the dragon without first completing the *Merchants' feud* quest? According to the previous explanation, all the quest points' success encounters can fire independently. The answer is that those encounters can have a condition that only allows the encounter to fire if a given quest point is enabled (there is also a counterpart condition requiring a quest point to succeed). Thus, the conversation with the dragon can be prevented until the *Merchants' feud* succeeds. Although this solution is not as elegant as desired, it works. A better solution is described in the Chapter 6 - Future Work and Conclusions .

3.2.8 - Commitment

In the dragon example of Figure 3.12, the PC can complete both the *Exterminate* and *Retrieve* branches of the quest by first retrieving the gemstone and then killing the dragon. The author may not want to allow the PC to succeed at both subquests. ScriptEase uses the concept of *commitment* to allow the author

to define a mutually exclusive set of quest points. A quest point can be *normal* (a dashed border) or *committing* (a solid border). By default a quest point is *normal*. When a *normal* quest point succeeds, it enables its successor quest points. However, it does not affect other quest points that are already enabled. In contrast, when a *committing* quest point (denoted by a solid border in the figures) succeeds it disables all quest points in the quest, except for the ones it enables.

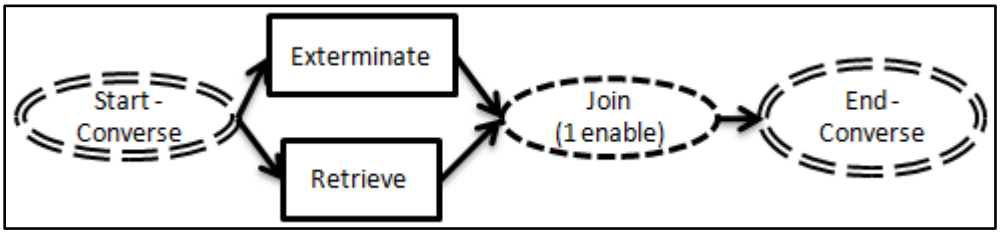


Figure 3.15 – *Do one of many* quest pattern with committing quest points

Figure 3.15 shows the dragon quest with two committing quest points. Figure 3.16 highlights the difference in what occurs when either type of quest point succeeds. As step one of Figure 3.16 depicts, when the *normal Start – Converse* quest point succeeds, it enables both the *Exterminate* and *Retrieve* subquests. Step two shows that when the *committing Retrieve* subquest succeeds it disables the *Exterminate* subquest and enables the *Join* quest point. The *Exterminate* subquest is permanently disabled as no quest point remains that could re-enable it. Thus, a committing quest point can be used to commit the PC to a specific branch of a quest. By itself, committing quest points only prevent other quest points from being enabled: the PC can return the gemstone to the dragon and then still kill the dragon, except now the *Exterminate* subquest is not

considered completed. However, by adding actions to those quest points' success encounters, the mutual exclusivity of the quest points can be further enforced. For example, when the PC gives the dragon the gemstone, the dragon could fly away as an action.

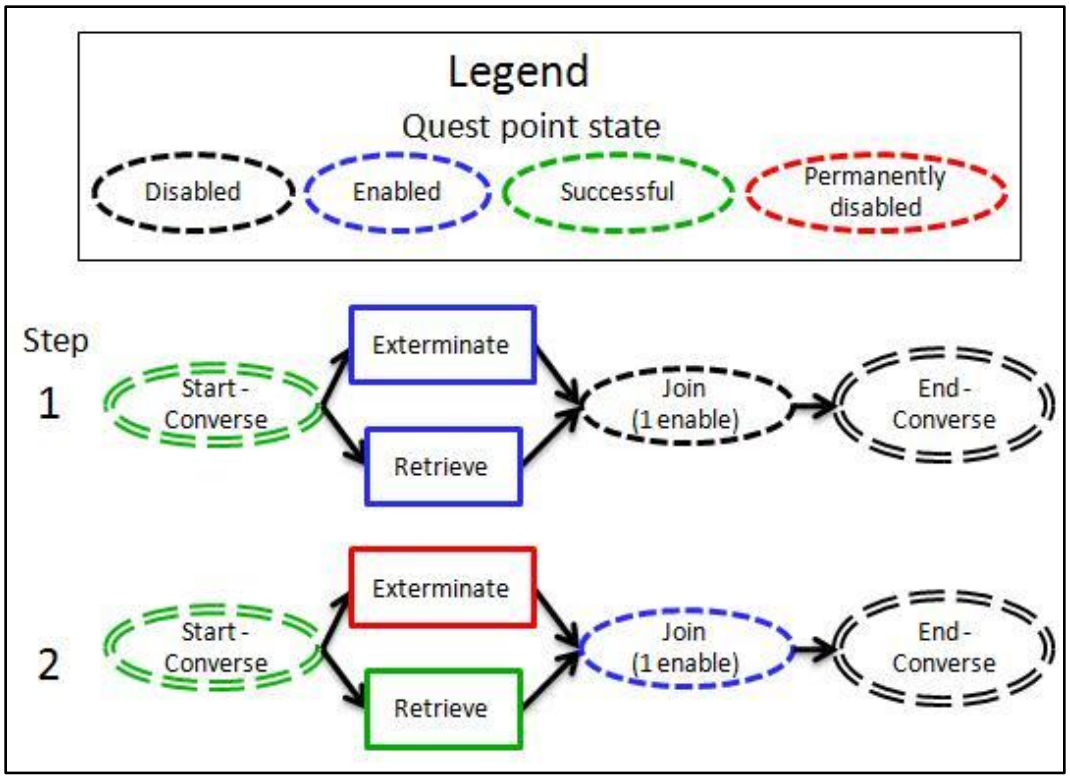


Figure 3.16 – Committing quest points are mutually exclusive

Committing quest points allow far greater control over the plot in other ways, namely expired branches and explicit failure. An expired branch is an optional, dead-end branch that is disabled at a certain point in the quest. Figure 3.10, which had an optional quest point of acquiring a sword, illustrates the need for supporting expired branches in the dragon quest. However, is it relevant to the

plot if the player acquires the sword after the dragon is defeated? No, especially if a journal entry explicitly states the sword should be used against the dragon. By this point in the quest, the relevance of the sword has expired. Adapting the *Kill* quest point into a committing quest point (Figure 3.17) solves this problem. If *Kill* succeeds before *Acquire* then *Acquire* is disabled due to the commitment of the *Kill*. Since *Acquire* is disabled it cannot be reached, and no journal entry will be given if the sword is acquired. The final *Converse* quest point still becomes enabled, allowing the quest to finish. Thus, the dead-end *Acquire* branch expires once the main task of *Kill* succeeds. This mechanism provides an elegant way of modeling hints or reminders given to the player.

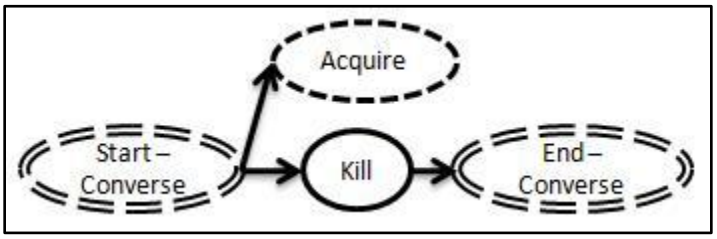


Figure 3.17 – Using commitment to cause a dead-end branch (*Acquire*) to expire if the main task (*Kill*) succeeds

Another use of commitment is to force explicit failure of a quest. In the dragon example, suppose if the player does not defeat the dragon by midnight then the dragon destroys the village, causing the player to fail the quest. The diagram for this quest is shown in Figure 3.18, in which the *Alarm* quest point represents the arrival of midnight. As described so far, the only way for a quest to fail is when it is impossible for the final quest point to succeed, which is implicit.

In general implicit failure is useful since it relieves the author from the responsibility of listing all of the ways a quest could fail. However, sometimes an author wants to force failure. Such is the case in this example. A committing dead-end quest point can be used to create an explicit failure event. Figure 3.19 demonstrates this procedure. In step 1, when *Alarm* succeeds, *Kill* is permanently disabled and *Fail* is enabled. Since there is still an enabled quest point (*Fail*) the quest as a whole does not fail yet. The *Fail* quest point is designed to succeed automatically upon becoming enabled (step 2), resulting in no enabled quest points remaining and the implicit failure of the quest as a whole.

It should be noted that the *Fail* quest point is completely redundant. If it was removed then the quest would fail as soon as *Alarm* succeeded. So why even use a *Fail* quest point? It serves as an explicit warning to the author that following this branch of the quest causes the whole quest to fail, as the branching structure of a quest is not obvious at a glance in ScriptEase (see Figure 3.20).

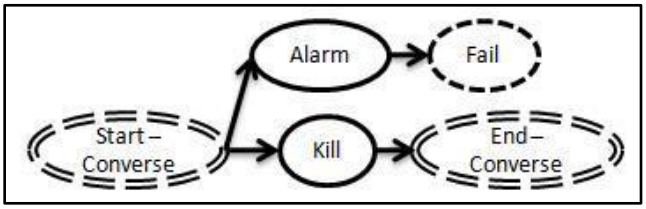


Figure 3.18 – An *Alarm* quest point as an explicit failure condition

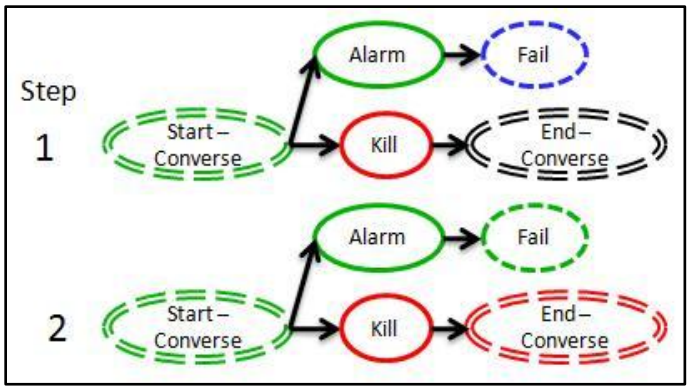


Figure 3.19 - How a dead-end committing branch can cause a quest to fail

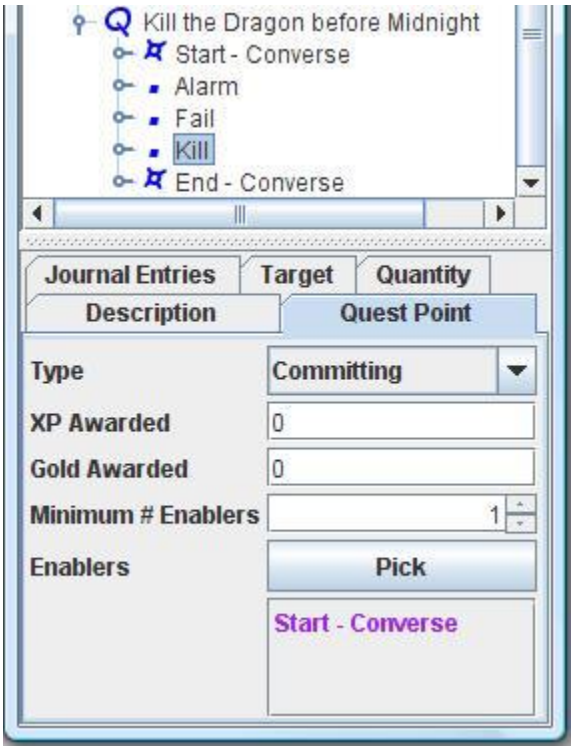


Figure 3.20 – For this quest the branching structure is not obvious and ScriptEase only shows a quest point's commitment when it is selected

3.3 - Improvements and Limitations

Quest patterns have evolved since the work of Curtis Onuczko [35]. My contributions to quest pattern mechanics are meta quest points and abandonable subquests. Abandonable subquests solved a problem of scope with subquests. Before a superquest could only enable a subquest, now a superquest can disable a subquest as well. Meta quest points have a more profound impact on the reusability of quest patterns and clarity of designer's intent. Before I conceived meta quest points, it was impossible to introduce structural variation into a single quest pattern. This prior limitation led to many similar patterns sharing the same intent. For example, the original quest catalogue contained three variations of patterns to retrieve one or more items, which have since been unified into a single *Retrieve* pattern. An analogous problem with the *Assassination* and *Assassination (with proof)* templates (the counterparts of quest patterns) occurs in the Plot Wizard. We can use the concept of a meta quest point to represent both templates in ScriptEase with a single *Assassinate* quest pattern using an *Obtain Proof* meta quest point containing a *Retrieve* subquest choice. Similar abstraction problems affect other tools for *NWN*. The meta quest point abstraction allows the catalogue of quest patterns to shrink while still providing the same expressive power.

Meta quest points do have limitations. Their lack of options impairs clarity. Normal quest points have options. For example, the *Converse* quest point has a *success dialogue line* option. However, the *Start* meta quest point, which can be set to a *Converse* quest point, should not have the same option. A *Start* can

also be set to an *Arrive* quest point, which does not have this option. Therefore, options for meta quest points are not appropriate in most circumstances. An example of a circumstance where meta quest point options are appropriate is the *Gain item* meta quest point. All of its intended quest points have an item option. In such cases meta quest point options are desirable.

However, if meta quest point options are supported, there is no simple mechanism for automatically linking the choices for the options of a meta quest point to choices for the options of a quest point, as was possible between quest patterns and quest points. A quest pattern contains its quest points, so a quest point option can be set in the scope of the quest pattern. However, a meta quest point does not contain its quest points. A meta quest point is a list of potential quest points with no scoping relationship. The result is that quest patterns that are entirely composed of meta quest points have no options, since there is no way for the meta quest points to use those options. This is the case with the *Retrieve* pattern, which is composed of four meta quest points: *Start*, *Gain item*, *Give item*, and *End*. In this case, it would be beneficial to have some way of declaring an item option in the *Gain Item* and *Give Item* meta quest points, since all of their intended quest points have an item option. The common item option could be propagated to the *Retrieve* pattern, saving the author from needing to choose the item multiple times.

The current quest model possesses three other limitations as well. Onuczko identified these limitations – loops, conflicting roles, and impossible

quest structure [35]. These still persist in the current iteration of ScriptEase. Rather than addressing these simple issues, effort was spent researching new techniques for designing patterns and studying the utility of ScriptEase and its pattern catalogue.

The first limitation involves an unintended relationship between quest points. There is no restriction to prevent quest points from forming a loop. Repeating a quest point should be a logical error. Instead, the model ignores the looping arc without informing the author, since a reached quest point cannot be re-enabled. The second limitation is the assignment of NPCs to contradictory roles. Frequently, quests require killing one NPC (the victim) and then reporting the act to another NPC (the employer). Nothing in the quest patterns prevents the same NPC from being selected to fulfill both roles. It is impossible to talk to a dead NPC, so that quest can never succeed. The third limitation is that the user may create a quest that is structurally impossible to complete – such as having no links to the final quest point – but the author will not be warned.

Two of these issues (loops and impossible quest structure) can be solved by running existing algorithms at quest instantiation time, so they are not interesting research issues. The remaining issue (conflicting roles) can be solved in a straightforward manner by allowing an author to specify a quest point at which an object's relevance expires and it cannot be used in the options of following quest points.

3.4 - Implementation

ScritpEase patterns have been described in this dissertation, but patterns rely upon the underlying ScriptEase implementation to generate code. In addition, the ScriptEase program requires programming code to construct and edit parse trees, generate properly formed scripts, and read/write ScriptEase objects from/to game modules.

ScriptEase is written in java so it can be easily ported between operating systems. The project is six years old and has evolved during my research. The graphical user interface, the ability to read and write to *NWN* modules, and the ability to generate scripts for encounters already existed. The ScriptEase Implementation Team wrote the java code for ScriptEase necessary for quest patterns [44]. About 2900 lines of java code were required to add the graphical user interface for quest patterns, the structure dictating the composition of the patterns, and the ability to generate scripts for quest patterns. This java code was written over a period of two years by the implementation team.

In addition to the challenge of integrating these 2900 lines of code into the rest of the ScriptEase source code, quest patterns raised further challenges. Unlike the other patterns used by ScriptEase, quest patterns use recursion. Subquests cause this recursion by having quests within quests. The recursion relationship becomes indirect when a meta quest point is set to a subquest. The relationship becomes a quest within a meta quest point within a quest.

Not only was java code necessary to implement quest patterns, but new atomic ScriptEase components were also written in *NWN*'s scripting language, NWScript. Scripting code was needed for both for the SciptEase patterns and to provide library functions for the quest system that uses those patterns.

ScriptEase installs four scripts that serve as function libraries and the runtime system for behaviours and quests. Specifically, there are two scripts that implement the runtime system for quests: one for the journal entries and one for the quest system. Both of these scripts were written by the ScriptEase Implementation Team [44]. Upgrading the journal system so it lists all entries for a quest rather than overwriting the last journal entry and allowing for gender-specific text substitution required 269 lines of NWScript code. The runtime system for quests that tracks the state of each quest and quest point as well as the transition between states required 881 lines of NWScript code. In total, the ScriptEase Implementation Team wrote 1150 lines of NWScript code to implement quest patterns in addition to the 2900 lines of Java code.

ScriptEase uses a hierarchy of patterns, which at the lowest level consists of actions, definitions, and events. When new patterns of these types were added new NWScript code was required. These new patterns and existing patterns were synthesized to create the quest catalogue described in Appendix A - Quest Catalogue. To create these new patterns, in addition to the ScriptEase patterns that I constructed, I wrote 452 lines of NWScript code for new actions, definitions and events. These lines of code controlled constructs such as the countdown timer

used in the *Race against time* pattern and such simple things as fetching and storing data. However, I spent far more effort on creating the quest patterns, quest points and meta quest points for the catalogue, out of encounters, actions, conditions, definitions and events, than on writing the NWScript that implements those patterns.

Chapter 4 - Quests in Commercial Video Games

Throughout this dissertation, I have stressed the seven qualities needed by a quest model and its tools: adaptability, clarity, ease-of-use, effectiveness, reusability, robustness, and scalability. The improvements made to quest patterns described in the previous chapter enhanced reusability and scalability. These improvements were used in a pair of studies to determine the usability of quest patterns.

4.1 - Previous Studies and New Questions

Three of the crucial qualities are easy to evaluate empirically: adaptability, ease-of-use, and effectiveness. Adaptability can be shown by using the quest model to represent the breadth of quests found in multiple commercial video games. Ease-of-use and effectiveness can be demonstrated by user studies. Though many of the related works do not provide empirical evidence supporting their tools, several studies provide such evidence for quest patterns and ScriptEase.

Three of these studies were performed before my work on quest patterns. The first study showed ScriptEase's ease-of-use through a user study in which high school English students used encounter patterns to create their own stories [7]. In the second study, Curtis Onuczko performed a case study that showed a catalogue of five quest patterns could represent nine quests found in a section of *Neverwinter Nights*' (*NWN*) official campaign story [35]. He also showed in a third study that

the structure of quest patterns was suitable for randomly generating quests. When such quests were complemented with manually-written dialogue, they were judged by participants of the user study to be as good as manually constructed quests.

However, three questions about ScriptEase and quest patterns still remain:

1. Are quest patterns adaptable enough to accurately represent all the quests found in a story-based, commercial video game?
2. Are quest patterns in ScriptEase more effective than manual scripting for programmers?
3. Can non-programmers use ScriptEase quest patterns?

The answer to the first two questions is *yes*, and the supporting empirical evidence is detailed in this chapter and the next. The answer to the third question is probably *yes* since it has been shown through user studies that non-programmers can use other ScriptEase patterns. In addition, students taking the CMPUT 250: Computers and Games course at the University of Alberta have successfully used a preliminary version of quest patterns. I focused on whether quest patterns provide a professional script programmer a more desirable alternative to manual plot scripting. The more difficult question of whether quest patterns are more effective than manual scripting was answered by a user study in Chapter 5 - ScriptEase versus NWScript, in which university students scripted quests using both ScriptEase and the native, text-based NWScript editor for *NWN*.

The question of adaptability of quests patterns for commercial video games was answered by examining all the quests found in the story-based, commercial video games *Elder Scrolls IV: Oblivion* (*Oblivion*) [16] and *Star Wars: Knights of the Old Republic* (*KOTOR*) [49]. I measured the compatibility of the quests of these games with ScriptEase's catalogue of quest patterns that was designed for use with *NWN*.

4.2 - Updating the Pattern Catalogue

No matter how easy-to-use a tool is, if it cannot perform the task required by its users then it will not be used. This principle also applies to quest patterns and ScriptEase. One of the most important aspects of using generative design patterns is the size and quality of the pattern catalogue. If the catalogue is too small then the users will spend too much time adapting and augmenting the patterns. If the catalogue is too big then it will be difficult to browse and users may end up selecting the wrong pattern or creating a redundant pattern when they cannot find the appropriate pattern. Both outcomes lead to excessive adaptation and augmentation. If the catalogue is of poor quality then the chosen pattern will need to be heavily adapted before being useful. Thus, it is important to have a catalogue that is the proper size and contains the most popular patterns – both simple and complex.

I spent considerable time on development, application, and evaluation of the pattern catalogue. The other automated scripting tools for *NWN* mentioned in

Chapter 2 - Related Work all had limited catalogues. Designing new patterns in ScriptEase is easy – often not requiring programming knowledge – allowing the quest catalogue to expand greatly. Curtis Onuczko's original catalogue of 5 quest patterns grew after his dissertation was finished. I began my thesis with the existing catalogue of 12 quest patterns and 20 quest point patterns, listed in Table 4.1. My first task was to redesign the quest pattern catalogue. These patterns were added subjectively initially, based on my experience with video game plot structures. New patterns were added and redundant ones were removed. Meta quest points were added to every quest pattern in the catalogue as well, resulting in the catalogue in Table 4.3.

For this study, the plots of *Oblivion* [16] and *KOTOR* [49] were analyzed. Metrics measured the effectiveness of the patterns. Then patterns were added, edited, or removed from the catalogue and the updated catalogue was remeasured. The summary of the changes in the catalogue for each game is shown in Table 4.2. The first number in each cell is the number of patterns in the catalogue after the catalogue was updated. The numbers in brackets reveals how many patterns were added and removed from the catalogue due to the analysis of that game. The details of these updates to the catalogue are described in Table 4.4 and Table 4.5, respectively. The catalogue resulting from both analyses is detailed in Appendix A - Quest Catalogue.

Quests (12 total)	Quest Points (20 total)
Any	Alarm
Converse – All	Acquire
Converse – Approach – Converse	Approach
Converse – Arrive – Converse	Area barrier
Converse – Kill – Converse	Arrive
Feud	Arson
Implicate	Automatic
Journey	Converse
Malign	Converse – take item
Phases	Convince
Placeholder	Fail
Wait	Join
	Kill
	Pay
	Place item
	Quest completed
	Rendezvous
	Take non-blueprint
	Trigger
	Verbal skill

Table 4.1 – Quest catalogue inherited from Curtis Onuczko

	Initial	Oblivion	KOTOR
Quests	26	33 (+7)	36 (+3)
Quest Points	30	46 (+17, -1)	49 (+3)
Meta Quest Points	17	17 (+1,-1)	17

Table 4.2 – The number of patterns in catalogue after analyzing each game

Quests (25 total, 14 new)	**Quest Points (30 total, 14 new)**	**Meta Quest Points (17 total, all new)**
Activate	Alarm	**Activate placeable**
Ambush	Acquire	**Any**
Area barrier	Approach	**Arrange meeting**
Assassinate	**Area barrier falls**	**Discover**
Convince	**Area barrier rises**	**End**
Do in any order	Arrive	**Escort begins**
Do one of many	Arson	**Escort ends**
Escort	Automatic	**Gain authority**
Expel	Converse	**Gain item**
Exterminate	Converse – give item	**Give item**
Feud	**Converse – has item**	**Hint**
Hunt	**Converse – take item**	**Hire**
Implicate	Converse – use skill	**Obtain proof**
Investigate	Fail	**Start**
Journey	**Filter class**	**Start ambush**
Pause	**Filter gender**	**Track**
Placeholder	**Filter level**	**Travel**
Race against time	**Filter race**	
Rally	Join	
Recovery expedition	Kill	
Rescue	Pay	
Retrieve	Quest completed	
Search	Place item	
Subquest chain	**Timer aborts**	
Wait	**Timer expires**	
	Timer starts	
	Trigger	
	Use placeable	
	Use item with placeable	
	Use spell on placeable	

Table 4.3 – Quest catalogue updated from my experience. Added patterns are bolded; the rest are from Table 4.1, with some renamings and deletions.

Additions

Quests (7)	Quest Points (16)	Meta Quest Points (1)
Backup plan Bodyguard Conditional reward Deadline Follow Gladiator Profit	Converse – exchange items Earn Equip Filter amount Leave area Murder NPC perceives NPC NPC perceives PC Open placeable PC dies Rest Status effect Steal Unequip Use item Use spell on NPC Wound	Negotiate

Deletions

Quests (0)	Quest Points (1)	Meta Quest Points (1)
	Converse – take item	Any

Edited

Quests (1)	Quest Points (3)	Meta Quest Points (8)
Investigate	Approach Converse – give items Timer expires	Activate Discover End Escort begins Escort ends Give item Hint Start

Table 4.4 – Quest catalogue updated from Oblivion, relative to Table 4.3

Additions

Quests (3)	Quest Points (3)	Meta Quest Points (0)
Advocate Backstory Tournament	Menu choice Minigame Party member	

Edited

Quests (6)	Quest Points (0)	Meta Quest Points (4)
Assassinate Convince Cooperate Expel Feud Search		End Escort begins Give item Start

Table 4.5 – Quest catalogue updated from KOTOR, relative to Table 4.4

Why not analyze *Neverwinter Nights'* (*NWN*) [32] main campaign as well? ScriptEase was designed with *NWN* in mind. A previous study [12] already measured the effectiveness of the encounter pattern catalogue in generating *NWN* main campaign stories. I expected the quest pattern catalogue to accurately reflect the quests used in *NWN*'s main campaign since ScriptEase was designed with that game in mind. Therefore, to test the effectiveness of our quest pattern catalogue, I selected different games for study: *Oblivion* and *KOTOR*. Both *Oblivion* and *KOTOR*, like *NWN*, are popular story-based games. Bioware Inc.'s [4] *KOTOR* won 126 awards since its release in 2003, and Bethesda Softworks' [3] *Oblivion* won 103 awards since its release in 2006. Like *NWN*, *Oblivion* provides a powerful toolset – reliant on scripting – and has an active online community making new game components and scenarios. These two companies are major North American developers of video role-playing games, a genre that I observed

frequently boasts about their intricate plots. *Oblivion* and *KOTOR* are representative of this genre, since they are critically-acclaimed for their complex plots. Representing the quests of both games using quest patterns would demonstrate that quest patterns are applicable to the complex stories of commercial video games.

No *Oblivion* or *KOTOR* code was generated for the quests, since ScriptEase only produces scripts for *NWN*. Thus, whether quest patterns are computationally feasible for scripting in these games is still unknown. However, each quest was successfully represented using quest patterns. Regrettably, there was not enough time to personally play each game enough to discover every nuance of their plot structure. Instead, the list of *Oblivion*'s quests and the details of each quest were derived from a pair of *walkthroughs* [17][34]: textual descriptions of quests from third-party sources. Similarly, another pair of walkthroughs was used for *KOTOR* [37][50]. These sources were chosen since they frequently refer to the exact journal entries and conversations from the game. This level of detail was necessary to accurately determine whether the quest could be represented with patterns.

Initially, each of *Oblivion*'s quests was represented by the most appropriate patterns, if they existed. After the initial evaluation of the quest catalogue, missing patterns were added to the catalogue to represent the concepts that could not be elegantly represented using the original pattern catalogue. The updated pattern catalogue was then evaluated again to measure the improvement. This process was

repeated on *KOTOR* using the catalogue updated from *Oblivion*. Thus, the pattern catalogue evolved through its use in *NWN*, *Oblivion*, and *KOTOR* – in that order.

The catalogue was updated in numerous ways. The simplest changes were adding, removing, or renaming a pattern. More intricate updates were performed as well. Some quest points were updated by changing their options or the encounter patterns they contained. Some meta quest points were updated by changing their list of intended quest points. Some updates changed the structure of a quest pattern by adding or removing quest points or branches. Other changes replaced a quest point with a meta quest point. The pattern changes were subjective in nature. I ensured the pattern made sense when considered in isolation and was general enough for reuse across games. In addition, the pattern was specific enough to address a need in the game. The improvement of these subjective changes to the patterns was measured objectively with metrics.

4.3 - Metrics

Four metrics were used to measure the effectiveness of the catalogue: usage, coverage, utility, and precision. These metrics were first introduced to measure encounter patterns used to recreate *NWN*'s official campaign story [12].

Usage measures how much of the pattern catalogue is used in a specific application, in this case a video game. *Usage* is expressed as a ratio of the number of patterns from the catalogue used in the application over the number of patterns in the catalogue. Thus, if an application uses 4 patterns out of a catalogue of 10

patterns, it has a usage of 0.4 or 40%. The number of instances of those patterns is irrelevant for usage. The score ranges from 0 (when none of the patterns from the catalogue are used) to 1 (when all of the catalogue's patterns are used). Higher usage is better since unused patterns are a hindrance to a user trying to find an appropriate pattern and contribute to unnecessary pattern creation, documentation, and maintenance costs. Since quest-related patterns can be divided by type – such as quests, quest points, and meta quest points – the usage metric can be applied to each type separately. Therefore, a game story could use 20% of a catalogue's quests, 60% of its quest points, and 30% of its meta quest points. The formal definition for usage is given in Figure 4.1.

$$PCat_t \stackrel{\text{def}}{=} \{patterns\ of\ type\ t\ in\ the\ catalogue\},$$

$$\text{where } t \in \begin{Bmatrix} quests\ (q), quest\ points\ (qp), \\ and\ meta\ quest\ points\ (mqp) \end{Bmatrix}$$

$$i_p \stackrel{\text{def}}{=} an\ unadapted\ instance\ of\ pattern\ p$$

$$\overline{i_p} \stackrel{\text{def}}{=} an\ adapted\ instance\ of\ pattern\ p \leftrightarrow$$

$$\exists\ adaptations\ a_1 \cdots a_n \ni \overline{i_p} = a_n \cdots a_1 i_p$$

$$IApp_t \stackrel{\text{def}}{=} \begin{Bmatrix} adapted\ instances\ of\ patterns\ of\ type\ t \\ used\ in\ the\ application \end{Bmatrix}$$

$$PApp_t \stackrel{\text{def}}{=} \{patterns\ of\ type\ t\ used\ in\ application\} \stackrel{\text{def}}{=} \{p \ni \overline{i_p} \in IApp_t\}$$

$$usage(catalogue, application)_t \stackrel{\text{def}}{=} \frac{|PCat_t \cap PApp_t|}{|PCat_t|}$$

Figure 4.1 – Formal definitions for usage

Utility measures how often each pattern is used. *Utility* is defined as the ratio of the number of instances of the pattern in the application over all patterns used in the application. The formal definitions for utility are given in Figure 4.2. For example if the *Retrieve* quest pattern has 12 instances in a game story that uses 4 different quest patterns then the *Retrieve* quest pattern has a utility of 3. The lowest score possible is 0, when the pattern is not used in a game story. Utility is an indicator of reusability. Higher utility is better since the effort needed to learn how to use the pattern and the cost of creating, testing, and maintaining the pattern is amortized over the number of uses. Like usage, utility can be applied separately to each type of pattern.

$$IAppCat_t \stackrel{\text{def}}{=} \{\overline{i_p} \in IApp_t \ni p \in PCat_t\}$$

$$utility(catalogue, application)_t \stackrel{\text{def}}{=} \frac{|IAppCat_t|}{|PCat_t \cap PApp_t|}$$

Figure 4.2 – Formal definitions for utility

Whether a catalogue contains the most appropriate patterns is measured by *coverage*. The exact definition of *coverage* (as formally stated in Figure 4.3) depends on the type of pattern. For meta quest points, coverage depends on the meta quest point's list of intended quest points. If a game's quest requires that a catalogue's meta quest point be adapted to an unintended quest point, then that game essentially requires a meta quest point that is not found in the catalogue. For example, *Acquire* is an intended quest point for the *Gain item* meta quest point, but the *Wound* quest point is not. Thus, a quest that requires a *Gain item* meta quest

point to be adapted to a *Wound* quest point requires a meta quest point that is not found in the catalogue. Coverage for meta quest points is the ratio of the number of meta quest point patterns in the game that are found in the catalogue over the total number of meta quest point patterns in the game. The highest coverage score is 1, meaning that all meta quest points used in the game came from the catalogue; Conversely, the lowest coverage score is 0, meaning none of the meta quest points in the game came from the catalogue. Therefore, higher coverage is better since user performs fewer adaptations to inappropriate meta quest points.

For a quest point, coverage is concerned with whether that quest point was intended to do what is needed by the story or must be replaced by a newly-created quest point. Coverage for quest points is defined analogously to coverage for meta quest points; coverage is the ratio of the number of quest point patterns in the game that are found in the catalogue over the total number of quest point patterns in the game. The range of quest point coverage scores and their implication is the same as coverage for meta quest points.

For quests, coverage considers how much structural adaptation is necessary to transform the quest pattern instance into the instance needed for a quest in the game's story. The possible quest adaptations are divided into three categories: trivial, minor, and major. The adaptation and their costs are shown in Table 4.6. The *adaptation cost* of a quest is the sum of the minor adaptations or infinite if there are any major adaptations. Trivial adaptations have no cost. A minor adaptation changes the length of a quest pattern; a major adaptation changes its

structure, such as branching. A trivial adaptation does not change the length or structure of a quest. The *n-coverage* of a story is defined as the ratio of the quests with an adaptation cost less than or equal to *n* divided by the number of quest patterns in the application. For example, suppose an application has 10 quests: 3 quests with an adaptation cost of 0, 5 with an adaptation cost of 1, 1 with an adaptation cost of 2, and 1 with an infinite adaptation cost. That story has a *0-coverage* of 3/10 = 0.3, and a *1-coverage* of (3+5)/10 = 0.8. The n-coverage of this application is 0.9 for n greater than or equal to 2. A high n-coverage value of an application for low *n*'s is better, since it means that an appropriate pattern with a low number of necessary adaptations is present in the catalogue.

$$\mathrm{a}_{\overline{i_p}} \stackrel{\text{def}}{=} a_1 \cdots a_n \ni \overline{i_p} = a_n \cdots a_1 i_p$$

$$\mathrm{TrivialAdapt} \stackrel{\text{def}}{=} \{trivial\ quest\ adaptations\}$$

$$\mathrm{MinorAdapt} \stackrel{\text{def}}{=} \{minor\ quest\ adaptations\}$$

$$\mathrm{MajorAdapt} \stackrel{\text{def}}{=} \{major\ quest\ adaptations\}$$

$$coverage(catalogue, application)_{mqp} \stackrel{\text{def}}{=} \frac{\left|PCat_{mqp} \cap PApp_{mqp}\right|}{\left|PApp_{mqp}\right|}$$

$$coverage(catalogue, application)_{qp} \stackrel{\text{def}}{=} \frac{\left|PCat_{qp} \cap PApp_{qp}\right|}{\left|PApp_{qp}\right|}$$

$$coverage(\overline{i_p} \in IAppCat_q) \stackrel{\text{def}}{=} \begin{cases} \infty, & if\ \mathrm{a}_{\overline{i_p}} \cap \mathrm{MajorAdapt} \neq \emptyset \\ \sum_{\mathrm{a} \in \mathrm{a}_{\overline{i_p}}} \begin{cases} 1, \mathrm{a} \in \mathrm{MinorAdapt} \\ 0, \mathrm{a} \in \mathrm{TrivialAdapt} \end{cases}, & \mathrm{otherwise} \end{cases}$$

$$coverage(catlogue, application, distance)_q \stackrel{\text{def}}{=} \frac{\sum_{\overline{i_p} \in IAppCat_q} \begin{cases} 1, if\ coverage(\overline{i_p}) \leq distance \\ 0, otherwise \end{cases}}{|IAppCat_t|}$$

Figure 4.3 – Formal definitions for coverage

Adaptation	Coverage Category	Precision Cost
Setting a meta quest point	Trivial	0
Replacing a quest point	Trivial	0.1
Chaining a quest point	Trivial	0
Intentionally changing the number of enables for a quest point	Trivial	0
Intentionally changing the number of branches	Trivial	0.1
Inserting a quest point linearly	Minor	0.2
Deleting a quest point linearly	Minor	0
Unintentionally adding branches	Major	0.3
Removing branches	Major	0.2
Adding a dead-end branch	Major	0.2
Unintentionally changing the enabler list	Major	0.4
Unintentionally changing number of enables required	Major	0.1
Changing the commitment of a quest point	Major	0.4

Table 4.6 – Types of Adaptation

While coverage measures the structural cost for adaptations, *precision* measures the amount of user effort needed to transform an initial quest pattern instance into the desired pattern instance. To be consistent with other metrics where higher is better, precision for a quest starts at 1 and is reduced by each type of adaptation performed, to a minimum of 0. Based on my experience with ScriptEase in constructing and using quests, I assigned an *precision cost* to each of the quest adaptations, as shown in Table 4.6. The more effort the user must expend

in time and thought, the more severe the precision cost. Though changing the commitment of a quest point is easy to do, a lot of thought is needed to anticipate the effects of the change, thus earning a severe precision cost. Every type of adaptation does not need to occur to reduce a quest's precision to 0, only a few complex adaptations are needed. As an example of precision, a quest pattern that requires setting meta quest points (cost 0), inserting one quest point in-line (cost 0.2), and adding a second branch (cost 0.3), has a precision 1 - 0 - 0.2 - 0.3 = 0.5. The precision for a game is the average precision of its quests. The formal definitions of precision are given in Figure 4.4.

$$cost(a) \stackrel{\text{def}}{=} \{the\ cost\ of\ adaptation\ a\}$$

$$cost(\overline{i_p}) \stackrel{\text{def}}{=} \sum_{k=1}^{n} cost(a_k)\ \ni \overline{i_p} = a_n \cdots a_1 i_p$$

$$precision(catalog, application)_q \stackrel{\text{def}}{=} \frac{\sum_{\overline{i_p} \in IApp_q} max\left(0, 1 - cost(\overline{i_p})\right)}{\left|IApp_q\right|}$$

Figure 4.4 – Formal definitions for precision

In Table 4.6, the precision cost for an adaptation does not always match the coverage category. Some trivial adaptations have a positive precision cost, while some minor adaptations have a zero precision cost. The reason for the apparent mismatch is that precision and coverage measure two distinct concepts: structural change and user effort. Even though deleting a quest point changes the quest's

structure, affecting coverage, it is an easy operation to perform, which does not affect precision.

4.4 - Results

These metrics were applied to the quests from *Oblivion* and *KOTOR*. *Oblivion* has 166 quests which are represented using hundreds of quest patterns (including subquests), hundreds of meta quest points, and over a thousand quest points. Using those counts as measures, *KOTOR* (with its 93 quests) is about half the size of *Oblivion*. The statistics of the quest, quest point, and meta quest point patterns – before and after the catalogue updates – are given in Table 4.8, Table 4.9, and Table 4.10 – respectively and summarized in Table 4.7. The first three rows of each table contain the raw data for the total number of patterns and instances used in each game. These data are used to calculate the usage and utility for the pattern type. Oblivion had 166 quests and KOTOR had 93. Each quest had precision and coverage scores that are not given in the tables. The summary for the metrics in Table 4.7 uses the average of the scores for each type of pattern. Since there are multiple definitions of coverage for quests, the 3-coverage is used in the average in Table 4.7.

Metric	Oblivion			KOTOR		
	Before	After	% Change	Before	After	% Change
Usage	0.85	0.88	+3%	0.56	0.58	+5%
Coverage	0.51	0.69	+34%	0.76	0.86	+13%
Precision	0.58	0.69	+19%	0.74	0.78	+5%
Utility	38	31	-19%	24	23	-4%

Table 4.7 - Summary of metric scores

	Oblivion		***KOTOR***	
	Before	**After**	**Before**	**After**
Patterns in catalogue	26	33	33	36
Patterns in game	21	28	14	17
Pattern instances in game	332	347	186	189
Usage	0.8077	0.8484	0.4242	0.4722
Utility	15.81	12.39	13.29	11.12
0-Coverage	0.1687	0.1687	0.3548	0.4408
1-Coverage	0.2771	0.4096	0.5161	0.6129
2-Coverage	0.3434	0.5000	0.6129	0.6989
3-Coverage	0.3976	0.5723	0.6344	0.7204
Precision	0.5801	0.6898	0.7365	0.7774

Table 4.8 – Quests in *Oblivion* and *KOTOR*

	Oblivion		***KOTOR***	
	Before	**After**	**Before**	**After**
Patterns in catalogue	30	46	46	49
Patterns in game	24	38	25	28
Pattern instances in game	1463	1523	732	746
Usage	0.8	0.8478	0.5435	0.5714
Utility	60.96	39.05	29.28	26.64
Coverage	0.6486	1	0.8929	1

Table 4.9 – Quest points in *Oblivion* and *KOTOR*

	Oblivion		***KOTOR***	
	Before	**After**	**Before**	**After**
Patterns in catalogue	17	17	17	17
Patterns in game	16	16	12	12
Pattern instances in game	619	674	368	387
Usage	0.9412	0.9412	0.7059	0.7059
Utility	38.69	42.13	30.67	32.25
Coverage	0.5000	0.5000	0.7500	0.8571

Table 4.10 – Meta quest points in *Oblivion* and *KOTOR*

The metrics mostly improved with the updates to the catalogue, which was expected. Quest patterns and quest point patterns improved the most, while meta quest points did not improve by much.

Ten new quest patterns were added to the catalogue by the end of the study. These new patterns are used in the second analysis of the games, improving usage. Since these new patterns better represented some quests in *Oblivion* and *KOTOR*,

coverage and precision increased. Utility dropped since the instances are now spread over more patterns in the updated catalogue.

The effects of adding new quest points were a little more subtle. More happened than a simple increase of nineteen new quest point patterns. Some new patterns were added to the catalogue for symmetry even though they were not used in the games. For example, since an *Unequip* quest point (when the PC removes a specific piece of equipment) was added to the catalogue, an *Equip* quest point was added as well for future use, even though it was not used. The addition allows the catalogue to anticipate future use rather than being updated ad hoc. Also, two quest points were replaced. In order to provide clearer and more powerful quest points, *Converse – give item* (to the PC) and *Converse – take item* (from the PC) were replaced with *Converse – give items* (a generic transfer with the function of both original give and take quest points) and *Converse – exchange items* (a transfer in which both characters give and receive items). The metrics' scores still changed in the same manner as they did for quests and for similar reasons. Usage and coverage went up while utility went down.

The metrics for meta quest points changed in different ways. A redundant meta quest point (*Any* which was redundant with the *Placeholder* quest) was removed and a new meta quest point (*Negotiate*) was added, keeping the overall number of meta quest points unchanged. Usage remained unchanged since the redundant quest point and the new meta quest point were both used in *Oblivion*. The substantial change came from the increase in intended choices for each meta

quest point. These allowed many costly *replacement* adaptations in the first analysis to become less costly *set* operations in the second analysis, increasing utility. Coverage did not change much in *Oblivion* since each of the eight meta quest points did not add enough new choices to cover everything needed. This was a subjective decision to prevent rarely-used choices from cluttering the menu for each meta quest point. For example, in one quest the PC gains an item by wounding another creature. However, a *Wound* quest point as a choice for a *Gain item* meta quest point seemed specific to only Oblivion and too rare to add the catalogue, as it would only be used in 1 of the 83 *Gain item* meta quest points used in *Oblivion*. This was an explicit trade-off in favour of utility over coverage. For *KOTOR* enough choices were added to improve coverage without adding anything too rare or specific to only that game.

More additions to the catalogue were made during the analysis of the first game (*Oblivion*) than the second (*KOTOR*). The second game only added 43% of the number of quest patterns that were added for the first game. The analogous percentage for quest points was 19%. The first game added a single meta quest point while the second game added none. There are three possibilities why this occurred:

1) *Oblivion* is larger or more creative game than *KOTOR*, so it possessed a greater variety of patterns due to size and variety.

2) *Oblivion* and *KOTOR* share many of the same patterns. If the order of analysis had been switched then *KOTOR* would have had as many new patterns as *Oblivion*.
3) The base catalogue was comprehensive enough to represent *KOTOR* without significant updates.

The first possibility seems unlikely since *KOTOR* uses almost as many patterns as *Oblivion*. *KOTOR* uses 61% of the number of quest patterns as used in *Oblivion*, with 74% and 75% as analogous percentages for quest points and meta quest points respectively. The percentages of pattern additions are far lower. Therefore, the smaller size of *KOTOR* is not likely to be responsible for the fewer number of additions to the catalogue.

The second possibility is also unlikely. *KOTOR* did not use many of the patterns that *Oblivion* added. Only one of the seven quest patterns added from *Oblivion* is found in *KOTOR* (*Conditional reward*). The one meta quest point *Oblivion* added is found extensively in *KOTOR* (*Negotiate*). Of the sixteen quest points originally found in *Oblivion* only four are found in *KOTOR* (*Wound*, *Open placeable, Leave area, and Filter amount*). Obviously, none of the patterns originally found in *KOTOR* were found in *Oblivion* or else they would have been found in *Oblivion* first.

From a gameplay perspective, it makes sense that these two games would use different plot elements; *Oblivion* and *KOTOR* possess different strengths as a

game. *Oblivion* emphasizes the environment (with a day-night cycle), the legal system of the world, and the artificial intelligence (AI) of its NPCs. Therefore, many of the plots focus on the environment (*Rest*, *Trigger*, *Alarm*, and *Pause* are used frequently) and NPC's observations (*NPC perceives NPC*, *Steal*, and *Murder* are present in *Oblivion* but not in *KOTOR*). The quests of *Oblivion* tend to be rather simple and linear, as the simple *Exterminate* and *Retrieve* quest patterns were the most commonly used. *KOTOR* focuses on detailed, non-linear quests, instead of the environment and AI. Many of *KOTOR*'s quests have two branches: one involving good actions and the other involving evil actions. Thus, the *Do One of Many* quest pattern was the most frequently used quest pattern. The potentially-lengthy dialogue-oriented *Investigate* and the simple combat-oriented *Exterminate* were tied for next most frequently used quest pattern.

The evidence supports the third possibility. Despite emphasizing different gameplay elements in the plot, *KOTOR*'s patterns were already represented in the catalogue since *KOTOR*'s initial scores for precision and coverage were almost always higher than *Oblivion*'s final scores. Therefore, the initial quest catalogue (as shown in Table 4.3) was suited for *KOTOR*, requiring few updates to accommodate *KOTOR*'s quests. Perhaps this is due to the fact the *NWN* and *KOTOR* were created by the same game studio, Bioware. Five of the nine core game designers for *NWN* worked on *KOTOR*.

4.5 - Conclusion

The results for the study are positive. For the case of *Oblivion* and *KOTOR*, the catalogue represented both games' complex quests while requiring only a few additions. Even though the initial catalogue was not designed with these games in mind, 29 new patterns needed to be added to an existing catalogue of 73, a 40% increase in size. These additions increased coverage for *Oblivion* between 43% and 48% for 1-coverage, 2-coverage, and 3-coverage, while 0-coverage was unchanged. In *KOTOR*, all the coverage measures improved between 13% and 25%. Both games benefitted from an increase in precision, 19% for *Oblivion* and 5% for *KOTOR*. *KOTOR* benefitted less since the initial catalogue already described its quests precisely without needing many updates, demonstrating the reusability of the catalogue. As well, *Oblivion* benefitted greatly from the catalogue updates, showing that the pattern catalogue is adaptable from game to game.

Normally, a high usage is good. However, a large difference in usage before and after the catalogue update is not, as it shows that many more new patterns were needed to represent the game's quests. However, adding the new patterns produced only a small change in usage (for each pattern type up to a 6% increase in *Oblivion* and up to a 12% increase in *KOTOR*). This result reveals that neither game required a drastic increase in the number patterns, of any type, to represent its quests more effectively.

However, the increase in coverage, utility, and usage came at a cost. The greater variety of quest and quest point patterns in the catalogue caused a drop in utility between 19% and 56%. Since *Oblivion* required more new patterns it had the greater drop in utility. Despite this drop, each quest pattern was still used at least an average of 12 times and an average of 26 times for quest points. The patterns were still being reused frequently, showing the pattern catalogue was not overly-specialized. Thus, the costs of expanding the pattern catalogue are outweighed by the benefits. Overall, this study supports the effectiveness of quest patterns in commercial, story-based video games.

Chapter 5 - ScriptEase versus NWScript

Since I claim that ScriptEase is easy-to-use and effective, it is natural to compare it to the original tool used for scripting in *NWN*, NWScript. While a previous study demonstrates that high school students are capable of using encounter patterns [7], this study compares the use of quest patterns versus manual scripting by university students to author stories in *NWN*. This study was approved by The Faculties of Arts, Science, and Law Research Ethics Board under application number 1956(CLG08-12-01).

There were two objectives for the study: determine which tool participants could use to produce scripts more effectively and which tool participants preferred. The participants were asked to script eight predesigned quests using both NWScript and ScriptEase (using the post-KOTOR catalogue from the previous chapter). Additionally, participants tested those scripts for correctness by playing their creations in *NWN*. A quest was considered *implemented* when a participant believed the scripts required by the instructions were complete and had started working on the next quest. It does not necessarily mean that the scripts were correct. To determine the effectiveness of a tool, I measured the number of quests implemented, the number of implemented quests that functioned correctly, and the amount of time needed to implement quests (which was recorded on the form in Appendix H - Quest Comparison). Preference for one tool over the other was determined by compiling questionnaire responses concerning each individual quest (Appendix H - Quest Comparison) and the overall experience (Appendix I -

Overall Tool Comparison). Each preference question used a 5-point scale. Each quest had a preference question concerning ease-of-use, speed, and ease-of-debugging. Overall preference had two questions: which tool is preferred for the study and which is preferred for future use. These two overall preference questions were aggregated to give a preference score for the tool.

5.1 - Method

This study used an incomplete repeated measures design [45]. The participants were randomly divided as evenly as possible into two groups: the group that used ScriptEase first and the group that used NWScript first. The study had four phases. The first phase was a demographics survey (which is found in Appendix D - Demographics Form) that gathered data such as the participant's age, gender, field of study, and prior use of ScriptEase and NWScript. For the second phase, each group used the initial tool. For the third phase each group used the other tool. The fourth phase consisted of the participants completing a questionnaire on their preference between the tested tools (found in Appendix I - Overall Tool Comparison).

The incomplete repeated measures design was selected due to the length of the study and the small number of participants. While this design does make the study lengthier by having each participant use both tools, it allows the participants to directly compare both tools, an economical use of the few participants. The repeated measures occurred during one session, as it seemed too unlikely that all

participants could attend a session for each tool. The six-hour length of the study session could have exhausted the participants, biasing their performance. This concern was addressed by providing a lunch with food and drink as well as balancing which tool was the first to be used across groups. The balancing and randomization of the order of tools also counters other biases, such as preferring the first tool used or being more effective with the last tool through experience. Due to the length of the experiment, participants were monetarily compensated upon signing the consent form (in Appendix C - Consent Form).

Participant recruitment focused on computing science students at the University of Alberta, though it was not limited to them. The participant briefing was sent out by email (found in Appendix B - Participant Briefing). The twenty-three participants who responded were assigned either January 17th or January 24th of 2009 as the day of their experimental session. The two groups were balanced by numbers with twelve for the earlier date and eleven for the later.

The participants were screened for familiarity with the necessary concepts. All but one of the twenty-three participants were computing science or engineering students at the University of Alberta; the one exception had graduated previously. The participants were required to know at least one C-like programming language, to shorten the time needed to learn NWScript. Of the 20 participants that completed the demographics form precisely enough to answer the question of years of programming experience (responses on the questionnaire like “a long time” were omitted), they had a mean of 5.3 years, with a median of 2.5, and range

of 1 to 30 years. Familiarity with video games was also required since this experience would reduce the time needed to test scripts in the game. We expected that participants were already biased towards NWScript since its C-like structure was more familiar to participants than the more abstract patterns of ScriptEase.

The participants were quite familiar with video games. All of the participants had played at least one role-playing video game (a genre known for elaborate plots), and 78% of the participants had played seven or more role-playing games. A video game course at the University of Alberta uses *NWN* and an older version of ScriptEase that uses a prototype version of quest patterns. Therefore, we expected some of the participants to be familiar with *NWN*, some with its scripting language of NWScript, and some with ScriptEase's patterns. In fact, 52% of the participants had played *NWN* before with only a single participant that completed its main campaign story. A smaller percentage, 21%, had previously used NWScript and 17% had used ScriptEase. The participants were familiar enough to know what capabilities to expect from the tools but not familiar enough to know the tested tools thoroughly.

The participants were instructed to script eight quests for a story in *NWN* with each tool (those instructions are found in Appendix G - Quest Instructions). Except for the quests, the story was otherwise complete: all required terrain, objects, and conversations were already present. I assumed that eight quests were more than a participant could implement in the 175 minutes allotted for each tool, but they were encouraged to implement all quests to add time pressure. The time

pressure replicates the task of an actual video game programmer. A tutorial with an example quest was provided with each tool (Appendix E - NWScript Tutorial and Appendix F - ScriptEase Tutorial). The quests were implemented in a specified order. The same order was used for each tool. After the same quest was implemented with each tool, participants marked which tool they preferred.

The quests used are typical of those found in story-based games. None of the quests' descriptions used the name of the ScriptEase quest pattern. Therefore, it was non-trivial for a participant to find the appropriate pattern. In addition, to represent the situation where no appropriate pattern exists in the catalogue, one of the quests had no corresponding quest pattern in the provided catalogue. The catalogue used in this study is the one that resulted from the previous case study, with one omitted quest pattern. This mirrors the real-world situation where an author will need to spend time finding the most appropriate pattern. After a time limit was reached for both tools, the participants completed an overall preference questionnaire. The eight quests listed below were completely independent of each other and varied in complexity:

1. Lighthouse Inspection – The PC is hired by an engineer to inspect the lighthouse and report back.
2. The Lost Diamond – The PC finds a treasure map which leads to a diamond hidden among ruins.

3. Dragon Disposal – A villager pleads for the PC to remove a young dragon threatening the village. The dragon can be killed or driven off by ringing the warning bell.
4. Save Jurenglade from the Undead – A farmer hires the PC to defeat the zombies attacking the village.
5. Medicine Run – A doctor orders the PC to fetch medicine from a merchant.
6. A Taxing Situation – A fisher wants the PC to frame the tax collector by planting contraband in his tax chest.
7. The Bandits' Hostage – The PC must save a young boy from bandits and return him to his mother.
8. Wolfbane – A farmer hires the PC to kill the pack of wolves that threaten her cows.

The order of quests was chosen to measure specific characteristics of the tool. The first three quests were the most important. The first quest was a simple quest with a linear progression of events. This tested the most basic use of ScriptEase – a default linear order of events. A linear order is also the easiest to implement through manual scripting. Thus, the first quest served as a learning quest. The second quest was also simple and linear, but ScriptEase lacked the corresponding quest pattern in its catalogue, forcing the participants to adapt a more inappropriate quest pattern. This tested the participants' ability to fill a gap in the pattern catalogue by making substantial adaptations using ScriptEase. The third

quest was simple but had two branches. This tested the usability of ScriptEase's branching quests.

The purpose of the remaining five quests was to determine if participants could implement more quests with one tool over the other. The fourth through seventh quests were linear (with some dead-end branches) but successively increased in length and complexity. The seventh quest was the longest and most difficult; it used a dead-end branch and was the only quest to use subquests (it had two). The eighth and final quest was simple and very similar to the fourth quest to help distinguish between those participants who could not implement the previous, most-difficult quest due to a lack of time and those who could not solve the scripting tasks.

5.2 - Results

There are several confounding variables that threaten an incomplete repeated measure design. For example, it is possible that what the participants learned while using the first tool could help them use the second tool. Another concern is whether the participants could be exhausted when using the second tool. However, a set of T-tests show that these two concerns did not significantly affect the study. Comparing the percentage of the quests implemented correctly, participants averaged 46.04% when using NWScript first and 49.24% when using NWScript second, with a *p*-value of 0.424, when asking whether the second value is significantly higher than the first. For ScriptEase the respective values are

80.00% first, 68.45% second, that produce a *p*-value of 0.265. The overall percentage of correctly implemented quests per participant was 64.91% for the first tool used (whether it was quest patterns or NWScript) and 51.16% for the second tool used, producing a *p*-value of 0.139. Thus, what the participants learned with one tool did not significantly help them use the other tool, and neither did exhaustion significantly affect the study.

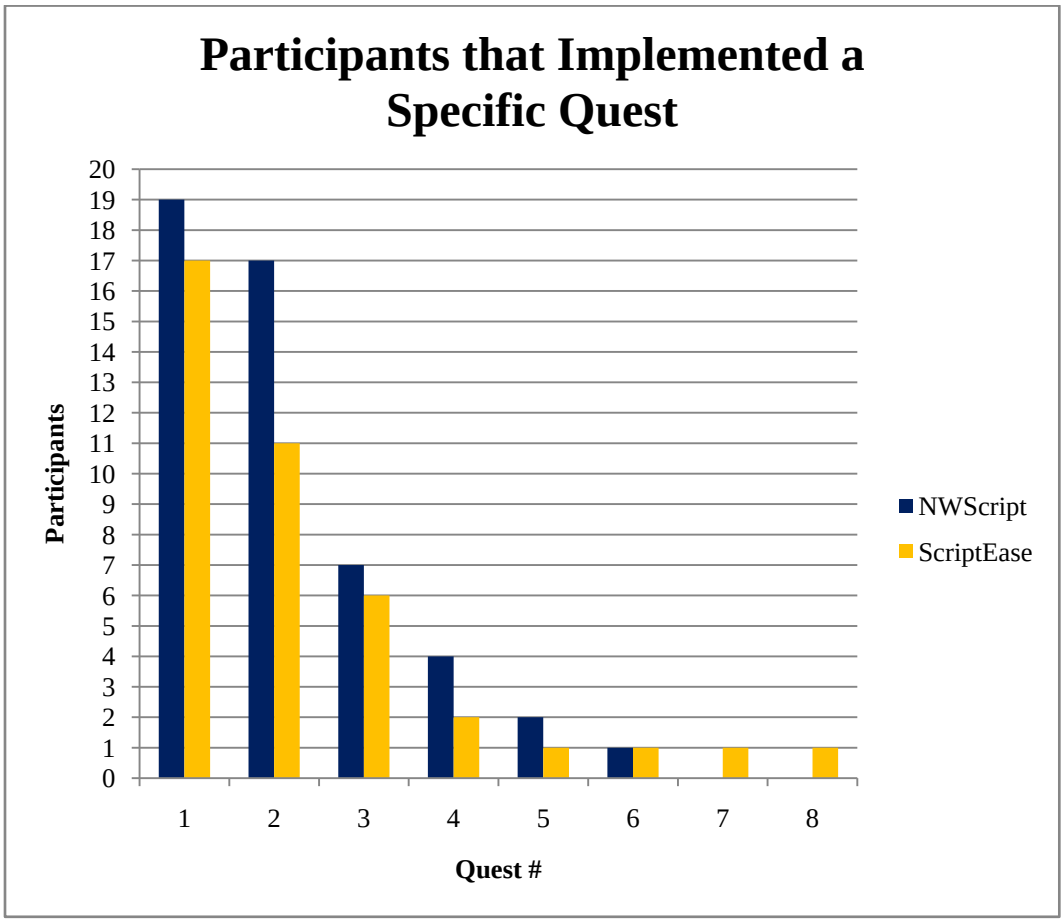


Figure 5.1 – Number of participants to implement a specific quest

However, the results of the study did not exactly match expectations, the main reason being that the study proved more difficult than predicted by the pilot

study. Figure 5.1 shows the number of participants who implemented each quest with each tool. Since the quests were completed in order from 1 to 8, the number of participants to implement each quest decreases. Four participants did not implement any quests within the time limit. Of the 23 participants, 6 implemented the third quest with ScriptEase and 7 did so with NWScript, which is far fewer than expected.

There is no statistical significance between the average number of quests implemented with each tool (1.74 for ScriptEase and 2.17 for NWScript, with a *p*-value of 0.194), which also defies my prediction that more participants would implement more quests using ScriptEase. However, as we will see later, participants correctly implemented more quests with ScriptEase.

However, Figure 5.1 does not tell the whole story about quest completion. Only one participant implemented three quests with both tools, though more participants implemented the first three quests with just one tool. The data suggests many participants were more productive with one tool than the other, though which tool was more productive varied from person to person. Figure 5.2 shows that only 3 of the 23 participants implemented the same number of quests with each tool, not including 4 participants who implemented 0 quests with each tool. The remaining 16 participants were more productive with one tool over the other. For example, there were 5 participants who implemented 2 more quests with NWScript than with ScriptEase.

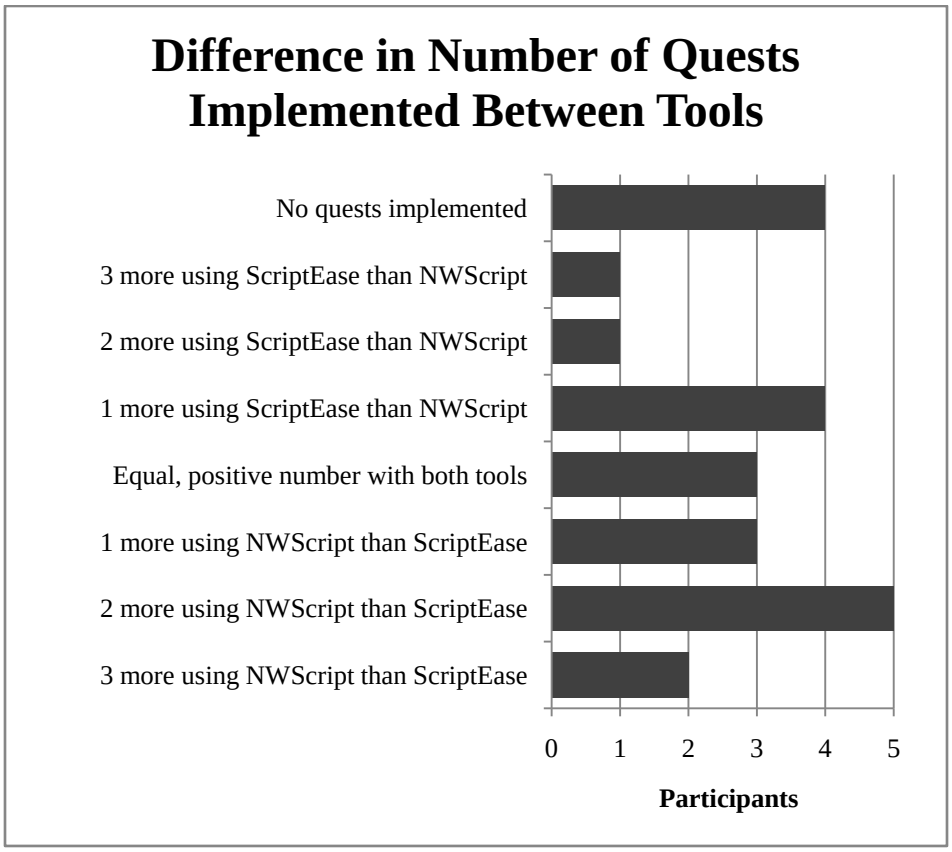


Figure 5.2 – Difference in Quests Implemented Between Tools

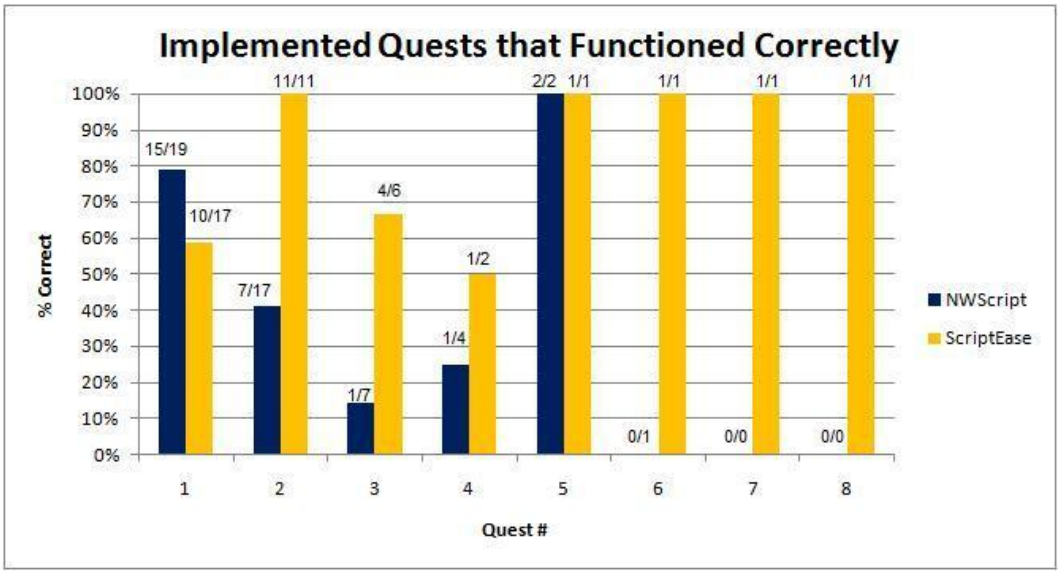


Figure 5.3 – Percentage of implemented quests functioning correctly

While no tool was more effective for completing more quests, there was a more effective tool for completing quests correctly. Participants were instructed to verify through testing that their quests met specifications. In addition, after the experiment I tested each quest to determine if it met the specifications. The correctness for each quest was binary: either a correctly fulfilled every requirement or it did not. Figure 5.3 shows that after the first quest the percentage of quests correctly-implemented with ScriptEase was at least as great NWScript's. However, for the first quest ScriptEase's scripts were less reliable than NWScript's, perhaps due to a steeper learning curve for ScriptEase.

In total, 30 of the 40 quests implemented using ScriptEase functioned correctly, but only 26 of the 50 implemented with NWScript functioned correctly. The number of correct quests per participant had a mean of 1.30 for ScriptEase and 1.13 for NWScript, which a T-test showed is not significantly different (*p*-value of 0.325). But, the percentage of correct quests per participant had a mean of 75.25% for ScriptEase and 47.89% for NWScript, which is significantly different. A one-tailed T-test, assuming unequal variance of the samples, produces a *p*-value of 0.0122, well within statistical significance at the 95th percentile. Therefore, scripts generated by ScriptEase are 1.57 times more reliable than scripts written in NWScript. For quests 2 and 3 specifically the multipliers are 2.43 and 4.67 respectively. Even with fewer samples the reliability difference between the tools for quests 2 and 3 is significant at the 95^{th} percentile; a one-tailed T-test for quests 2 and 3 produced *p*-values of 0.0001021 and 0.03482 respectively. No tool was

statistically more reliable for quest 1. The most frequent bugs found in the quests scripted using NWScript originated from quest events being completed in an unintended order.

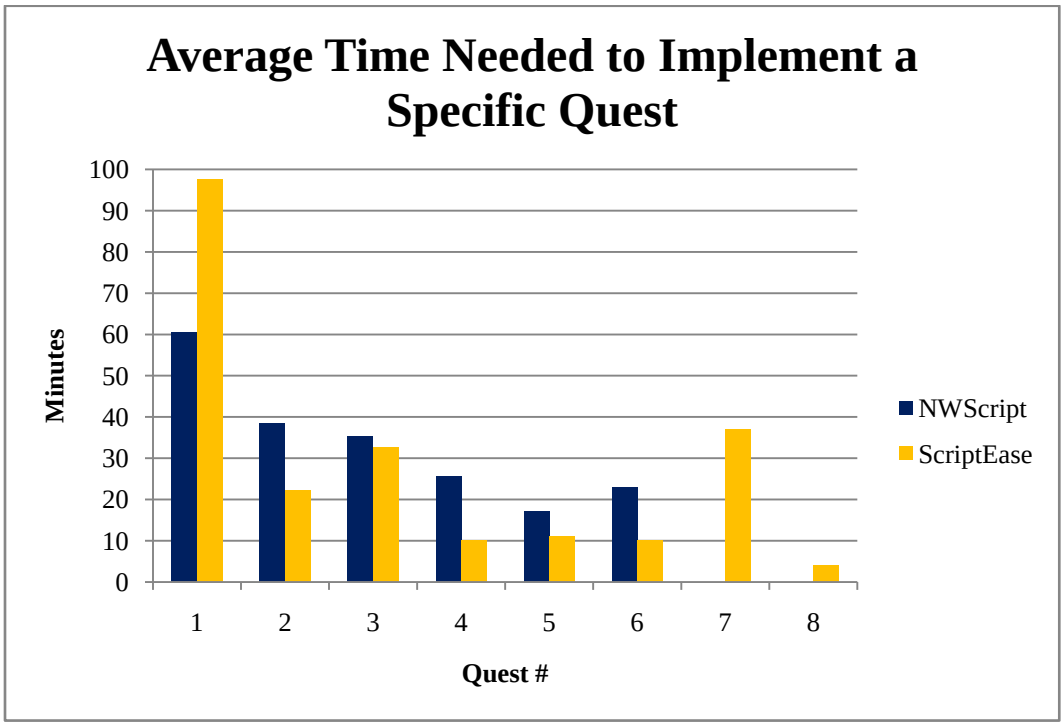


Figure 5.4 – Average minutes needed to implement an individual quest

The time needed to implement the quests revealed mixed results. Intuitively, Figure 5.4 shows that after a sizeable learning curve ScriptEase was quicker to use. However, this assertion cannot be supported statistically. I consider the time spent on the tutorial in addition to the time used to implement quest one as the learning curve for each technique. The mean time for this learning curve was 129 minutes for ScriptEase, compared to 90 minutes for NWScript. Using a one-tailed, unequal variance T-test – this difference is statistically significant at the

95th percentile with a *p*-value of 0.000394. On a quest-by-quest basis, NWScript was 61% faster for quest 1 with a *p*-value of 0.000685, ScriptEase was 42% faster for quest 2 with a *p*-value of 0.00384, and no tool was significantly faster for quest 3 (the *p*-value was 0.379). Further quests were not analyzed since so few participants implemented those quests. Even though NWScript was quicker to learn, it did not result in quicker times for later quests.

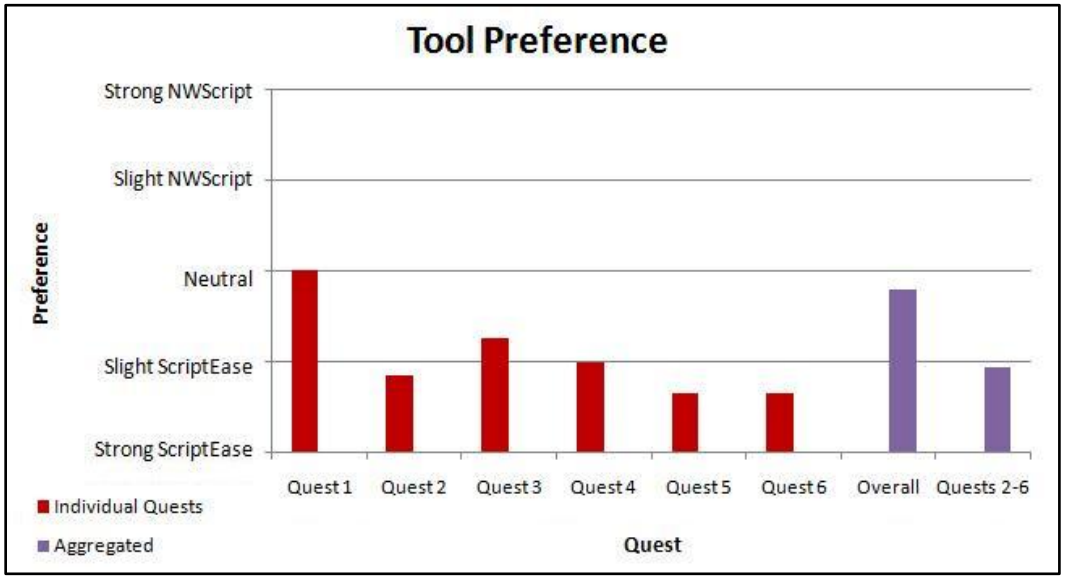


Figure 5.5 – Tool preference for individual and aggregated quests

The results for tool preference were somewhat positive. Z-tests were used for calculating preference, with a hypothesized population mean of neutral preference and using the sample's standard deviation. As shown in Figure 5.5 and verified by two-tailed Z-tests, participants were neutral about tool preference for quest 1 (*p*-value of 0.959), the learning quest, and overall (*p*-value of 0.486). However, after the first quest they significantly preferred ScriptEase. The one-

tailedZ-test on the preferences scores for quests 2 through 6 shows ScriptEase was preferred at the 95th percentile with a *p*-value of 2.042E-12.

The results for the reliability test were expected but the speed and preference tests' results were not. The expectation was that the participants would overall prefer to use ScriptEase and would implement more quests with it. The comments of the participants identified two factors that could have prevented that result. The first factor was the number of critical bugs in ScriptEase. Several of these bugs could only be fixed by restarting the program:

- a malfunctioning undo/redo operation,
- a memory leak for text input that rendered ScriptEase unusably slow after a few hundred typed words, and
- the inability to remove unwanted scripts from game objects.

Since ScriptEase was preferred after the first quest, it is possible the participants learned to avoid or work-around the bugs. Though NWScript was not free of bugs, they were fewer in number and less severe than ScriptEase's. However, participants commented that NWScript's interface was awkward and slow-to-navigate, preventing NWScript from achieving a better preference score.

The second factor was the length of the study; the participants commented that they only felt comfortable with the tools near the end of the allotted time. The study should have been longer to reduce the impact of the learning curve and allow for more data to be gathered. A longer study could have been more favourable to

ScriptEase; the data shows that participants preferred ScriptEase for the later quests, once they became familiar with its interface. Though ScriptEase had a longer learning curve, it was at least as fast as NWScript for later quests.

There was a third factor that strongly influenced the result that more quests were implemented using NWScript. Participants were often “fooled” into thinking that they had finished a quest with NWScript, even though it did not meet the specifications. Therefore, they proceeded to the next quest. This allowed them to implement more quests with NWScript. If an automatic verifier checked each implemented quest then it would have caught the bugs in the quests, slowing the development of the buggy NWScript quests. In this case, the number of implemented quests would have been reduced and the percentage of correct quests would have increased. Since using the respective tool, NWScript or ScriptEase, for debugging was part of the test, no such verifier was provided. The fact ScriptEase provides automated script generation to prevent many of these bugs indicates ScriptEase is more robust.

5.3 - Conclusion

The quest patterns of ScriptEase were more effective than the manual scripting of NWScript in several ways. ScriptEase was significantly preferred after the learning phase (completion of the tutorial and first quest). Significant results favouring ScriptEase were found in quest reliability, both overall and for quests 2 and 3 specifically. The data showed that overall ScriptEase’s quest patterns were

1.57 times more reliable (when correctness is binary) than NWScript's. More specifically, when an appropriate pattern was not available for a quest, ScriptEase was 2.43 times more reliable and 42% faster to script with than NWScript, supporting that ScriptEase's patterns are easily adaptable. Another impressive result is that ScriptEase was 4.67 times more reliable for the quest involving multiple branches, showing that ScriptEase's automated management of quest progression simplifies the process of creating more complex quests. In contrast, participants implemented insignificantly more quests using NWScript (the T-test yielded a *p*-value of 0.194); however, they frequently suffered from bugs caused by completing the quest's events in an unintended order.

For a commercial video game, more reliable scripts would reduce the cost of testing, especially for story-based games that frequently use branching plots (such as *Star Wars: Knights of the Old Republic* used in the next study). Therefore, the superior reliability of and preference for quest patterns supports the assertion that they are more effective than manual scripting for use in video games.

Chapter 6 - Future Work and Conclusions

6.1 - Summary

The multibillion dollar commercial video game industry needs better models and tools for scripting plots. I have described seven qualities that such models and tools should possess. Among all the techniques developed to relieve the plot scripting bottleneck, only the quest pattern model and the ScriptEase tool fulfill all seven criteria. They do so in the following ways:

- *Adaptability* through an easily-augmentable catalogue of patterns, which demonstrably represent the diverse quests of the complex, story-based, commercial video games *Elder Scrolls IV: Oblivion* (*Oblivion*) and *Star Wars: Knights of the Old Republic* (*KOTOR)*
- *Clarity* provided by using plain-English terminology instead of a programming language and by using abstraction so details are hidden from the initial view
- *Ease-of-use* by not requiring programming experience
- *Effectiveness* from automatically-generated scripts that use reliable templates, preventing many mundane errors that afflict manual scripting
- *Reusability* from a pattern catalogue, in which each pattern can be reused multiple times within a game and across multiple games

- *Robustness* through a quest model that tracks whether each quest point is enabled, fired, succeeded, or failed – which accommodates actions that would normally interrupt the intended progression of a quest
- *Scalability* by representing the relationship between plot elements using enabler links, subquests, and *Quest completed* quest points

Curtis Onuczko originally developed quest patterns [35], but I have improved the quest model relative to two criteria (reusability and scalability) and demonstrated the model's strength relative to two other criteria (adaptability and effectiveness) with a user study and a case study.

I have enhanced the reusability of quest patterns by creating meta quest points. A designer can now specify the intent for a quest point while allowing an author to choose a specific quest point matching the context of the story. This mechanism allows several similar quest patterns to be unified into a single pattern, as was the case when three fetch-and-deliver quest patterns in Onuczko's catalogue were unified into a single *Retrieve* quest pattern. Additionally, the large variety of beginnings (conversation, observation, acquisition, etc...) and endings (delivery, reward, arrival, etc...) for a quest can now be represented through *Start* and *End* meta quest points, rather than using separate quest patterns for each combination of beginning and ending. This new level of abstraction allows a small catalogue of 36 quest patterns, 49 quest point patterns, and 17 meta quest point patterns to represent 169 commercial quests by using 536 quest pattern

instances, 2269 quest point instances, and 1061 meta quest point instances. Without meta quest points, representing all those quests would require hundreds of additional quest patterns in the catalogue or require the author to perform thousands of adaptations to the pattern instances.

By using enabler links, subquests, and *Quest completed* quest points – quest patterns could already scale to represent complex plots. However, I further improved scalability by creating abandonable subquests. Abandonable subquests add a third outcome for a quest, abandonment, in addition to the existing outcomes of success and failure. This additional outcome allows for more branching quests in which a user can complete a quest using a multitude of methods, each method being represented as a subquest. Once the player completes one of those subquests, the rest of the subquests can expire, which was not possible in the previous quest model. In the previous model, the journal entries for the now-redundant subquests would remain open, misleading the player into believing that completing those subquests would further the plot. These branching quests are becoming increasingly common as more commercial games are emphasizing moral choices (like in *KOTOR*) or creative solutions to problems (like in *Vampire: the Masquerade – Bloodlines*). Such games can benefit from the subquest control provided by abanondable subquests.

I have demonstrated the effectiveness and adaptability of quest patterns by conducting a pair of studies. The adaptability of quest patterns was demonstrated by a case study of two popular, story-based video games: *Oblivion* and *KOTOR*.

The metrics of usage, utility, coverage, and precision were used to measure the effectiveness of the pattern catalogue, both before and after the catalogue was updated with the patterns found in each game. The updates increased the coverage scores between 0% and 48% in *Oblivion* and between 12% and 25% in *KOTOR*. Precision increased by 19% for *Oblivion* and 6% for *KOTOR*. Since the initial precision for *KOTOR* was already high at 0.7365, *KOTOR* did not benefit much from the updates since the catalogue already precisely represented the game. The additions to the catalogue did cause up to a 6% increase in usage for *Oblivion* and up to a 12% increase in usage for *KOTOR*. This slight increase is better than a larger increase, since a slight increase shows that only a few new patterns needed to be added to the catalogue to completely represent the game's quests. Overall, *Oblivion* best demonstrated the catalogue's adaptability, and *KOTOR* best demonstrated the catalogue's reusability. However, adding these new patterns caused a decrease in utility, the worst case being a 56% drop in utility for quest point patterns in *Oblivion*. However, even in that worst case, any quest point was still used on average 24 times in *Oblivion*, showing the patterns were highly reusable.

Together, these metrics show that quest patterns are adaptable enough to use in the complex plots of these commercial video games; this result hints that many story-based games, which have less complex plots, can benefit from quest patterns. Currently in video game development, confusion can arise between an author (who writes an intricate plot) and a script programmer (who implements

that plot). This confusion could be avoided by using the simple and expressive quest pattern model.

The other study was a user study in which twenty-three participants compared the automated scripting used by ScriptEase's quest patterns against the manual scripting used by NWScript in *Neverwinter Nights* (*NWN*). The participants were programmers, whom I believe were biased towards manual scripting since they were already familiar with a similar programming language. However, this study showed that quest patterns were significantly more effective in several ways. After the first quest, used for learning, ScriptEase was preferred. The quests scripted using quest patterns were more reliable overall and also for quests 2 and 3 specifically. Quest reliability was binary: either a quest correctly met all of the specifications or it did not. Overall, quest patterns were 1.57 times more reliable than those scripted with NWScript. That multiple became 4.67 when participants scripted a quest with two branches (quest 3). Even when the appropriate pattern for a quest was omitted from the catalogue (quest 2), quest patterns were 2.43 times more reliable and 42% faster to script than NWScript, which shows that quest patterns are easily adaptable. Altogether, this study shows that quest patterns are usable and more effective than the alternative manual scripting language.

6.2 - Future Work

The improvement and application of the quest pattern model answers some questions and raises others for future research:

- Since generative design patterns are applicable to video games, could they be applied to other tasks in which similar pieces of scripting code are written repeatedly, such as webpage programming?
- Now that we know that quest patterns are more efficient, more robust, easier-to-use, and preferable to manual scripting – another question is whether non-programmers can use them effectively; anecdotal evidence indicates the answer is yes, but a user study should be conducted.
- Since *KOTOR* was represented with only a small addition to the catalogue, is there a set of patterns common to all (or at least to the majority of) story-based games?
- Meta quest points have a list of quest points used for specialization. How large should that list be? How many entries would overwhelm an author?

On a more narrow scope, some of the quest points of the catalogue provide an opportunity for new research into encounter patterns, which were beyond the scope of my thesis. Some quest points serve as guards (like the *Quest completed* quest point) to other quest points. This is not the right layer of abstraction; such a guarding quest point should instead be a condition for the success encounter of the quest point it guards. But, making such conditions (and their associated

definitions) as easily interchangeable as quest points would require changing the model for encounters. Thus, the results of this research raise new questions for both the narrow scope of improving encounter patterns and the grand scope of applications for generative design patterns.

6.3 - Conclusions

My research makes several contributions; my creations of meta quest points and abandonable subquests add to the expressive power of quest patterns. The studies of quests in commercial video games and the ability of programmers to use quest patterns show that ScriptEase and quest patterns provide a desirable alternative to manual scripting in commercial, story-based video games. This research can help alleviate the bottleneck of plot scripting that afflicts modern video game development, for both commercial and educational purposes. Games with more complex plots can now be scripted more reliably, reducing development costs and allowing an author more direct control over the product. The pervasive, multibillion-dollar video game industry can benefit from this research.

Bibliography

[1] Alice. http://www.alice.org/.

[2] America's Army. United States Army. http://www.americasarmy.com/.

[3] Bethesda Softworks. http://www.bethsoft.com/eng/index.php.

[4] Bioware Corp. http://www.bioware.com/.

[5] C. Brom and A. Abonyi. Petri Nets for Game Plot. In Proceedings of the Society for the Study of Artificial Intelligence and the Simulation of Behaviour (2006). http://artemis.ms.mff.cuni.cz/main/papers/IVE-dramamanager-2006.pdf.

[6] S. Butts. Empire: Total War - The Post-Mortem. IGN. March 13, 2009. http://uk.pc.ign.com/articles/962/962469p1.html.

[7] M. Carbonaro, M. Cutumisu, H. Duff, S. Gillis, C. Onuczko, J. Siegel, J. Schaeffer, A. Schumacher, D. Szafron, and K. Waugh. Interactive Story Authoring: A Viable Form of Creative Expression for the Classroom. Computers and Education, 51(2): 687 – 707, September 2008.

[8] M. Cavazza, F. Charles, and S. Mead. Sex, Lies, and Video Games: an Interactive Storytelling Prototype. University of Teesside, 2002. www.scm.tees.ac.uk/users/f.charles/publications/conferences/2002/aaai2002.pdf.

[9] M. Cieply. Batman's 'Dark Knight' Sets Weekend Record. New York Times. July 21st, 2008. http://www.nytimes.com/2008/07/21/movies/20cnd-batman.html.

[10] Cost of making games set to soar. British Broadcasting Corporation. November 17th, 2005. http://news.bbc.co.uk/2/hi/technology/4442346.stm.

[11] M. Cutumisu. Using Behavior Patterns to Generate Scripts for Computer Role-Playing Games. Doctorate of Philosophy thesis. University of Alberta, 2009.

[12] M. Cutumisu, C. Onuczko, D. Szafron, J. Schaeffer, M. McNaughton, T. Roy, J. Siegel, and M. Carbonaro. Evaluating Pattern Catalogs - The Video games Experience. In proceedings of the 28th International Conference on Software Engineering (May 2006): 132-141.

[13] M. Cutumisu, D. Szafron, M. Bowling, and R. S. Sutton. Agent Learning using Action-Dependent Learning Rates in Computer Role-Playing Games. In Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (Palo Alto, USA, October 2008): 22-29.

[14] M. Cutumisu, D. Szafron, J. Schaeffer, M. McNaughton, T. Roy, C. Onuczko, and C. Mike. Generating Ambient Behaviors in Computer Role-Playing Games. IEEE Intelligent Systems, 21(5): 19-27, Sept./Oct. 2006.

[15] Defense of the Ancients. Modification of Warcraft III: Reign of Chaos. Blizzard Entertainment. Vivendi. http://www.dota-allstars.com/.

[16] Elder Scrolls IV: Oblivion. Bethesda Game Studios. Bethesda Softworks and Take-Two Interactive. http://www.elderscrolls.com/games/oblivion_overview.htm.

[17] The Elder Scrolls IV: Oblivion – Walkthrough. IGN and Gamespy. http://faqs.ign.com/articles/699/699097p1.html.

[18] Empire: Total War. Creative Assembly. Sega. http://www.totalwar.com/empire/index.php?t=EnglishUSA.

[19] Entertainment Software Association. http://www.theesa.com/facts/index.asp.

[20] G. Freytag. Freytag's Technique of the Drama: An Exposition of Dramatic Composition and Art. Scott, Foresman, Chicago, 1900.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1994.

[22] Goldeneye 007. Rare. Nintendo.

[23] Grand Theft Auto: San Andreas. Rockstar North. http://www.rockstargames.com/sanandreas/.

[24] Half-Life 2. Valve Corporation. Sierra. www.half-life2.com.

[25] S. Hillis. Microsoft says "Halo" 1st-week sales were $300 mln. Reuters UK. October 4th, 2007. http://uk.reuters.com/article/technologyNews/idUKN0438777720071005

[26] Homeworld 2. Relic Entertainment. Sierra. http://www.relic.com/games/.

[27] Left 4 Dead. Turtle Rock Studios. Valve Corporation. http://www.l4d.com/.

[28] Lilac Soul's NWN Script Generator V2.3. http://nwvault.ign.com/View.php?view=Other.Detail&id=625.

[29] M. MacNaughton, M. Cutumisu, D. Szafron, J. Schaeffer, J. Redford, and D. Parker. ScriptEase: Generative Design Patterns for Computer Role-Playing Games, 19th IEEE International Conference on Automated Software Engineering (September 2004, Linz, Austria): 88-99.

[30] M. Mateas and A. Stern. Façade: An Experiment in Building a Fully-Realized Interactive Drama. In Proceedings of the Game Developers Conference: Game Design Track, San Jose, 2003. http://www.interactivestory.net/papers/MateasSternGDC03.pdf.

[31] The Movie Industry Vs. the Gaming Industry. Associated Content. September 19th, 2008. http://www.associatedcontent.com/article/1015720/the_movie_industry_vs_the_gaming_industry.html?cat=19.

[32] Neverwinter Nights. BioWare Corp. Infogrames. http://nwn.bioware.com/.

[33] The Numbers. http://www.thenumbers.com/movies/series/JamesBond.php.

[34] Oblivion: Quests. UESPWiki. http://www.uesp.net/wiki/Oblivion:Quests.

[35] C. Onuczko. Quest Patterns in Computer Role-Playing Games. Master's Thesis. University of Alberta, 2007.

[36] C. Onuczko, D. Szafron, and J. Schaeffer. Stop Getting Side-Tracked by Side-Quests. In AI Game Programming Wisdom 4, Editor S Rabin. Charles River Media (2008): 513-528.

[37] J. Patterson. Star Wars Knights of the Old Republic FAQ/Walthrough. http://www.gamefaqs.com/computer/doswin/file/516675/29302.

[38] Plot Diagram for the Three Little Pigs. http://206.110.20.121/~schoenfe/Homework/SeptemberWork/Plot%20Diagram-3%20Little%20Pigs.doc.

[39] V. Propp. Morphology of the Folktale. University of Texas Press, Austin, TX, 1968.

[40] Rare. Microsoft Game Studios. http://www.rareware.com/company/press-microsoft1.html.

[41] R. Rosas, M. Nussbaum, P. Cumsille, V. Marianov, M. Correa, P. Flores, V. Grau, F. Lagos, X. López, V. López, P. Rodriguez, and M. Salinas. Beyond nintendo: design and assessment of educational video games for first and second grade students. Computers and Education, 40:71–94, 2003.

[42] W. Saar. Quest System for Massive Multiplayer Online Role-Playing Games. Master's Thesis. Royal Institute of Technology, 2004. http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2004/rapporter04/saar_william_04085.pdf.

[43] Scratch. http://scratch.mit.edu/.

[44] ScriptEase Implementation Team. http://www.cs.ualberta.ca/~script/.

[45] J. Shaughnessy, E. Zechmeister, and J. Zechmeister. Research Methods in Psychology. McGraw-Hill, New York City, 2006.

[46] Sid Meier's Civilization IV. Firaxis Games. Take-Two Interactive. http://www.2kgames.com/civ4/.

[47] J. Siegel and D. Szafron. Dialogue Patterns - A Visual Language For Dynamic Dialogue, Journal of Visual Languages and Computing, In Press 2009, 27 pages.

[48] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma. Adaptive game AI with dynamic scripting. Machine Learning, 63(3):217-248, 2006.

[49] Star Wars: Knights of the Old Republic. Bioware. LucasArts. http://www.lucasarts.com/index.htmls.

[50] Star Wars: KOTOR @ GameBanshee. http://www.gamebanshee.com/starwarskotor/walkthrough.php.

[51] Starcraft. Blizzard Entertainment. Vivendi. http://www.blizzard.com/us/starcraft/.

[52] A. Sullivan, S. Chen, and M. Mateas. Integrating Drama Management into an Adventure Game. In Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (Palo Alto, 2008). www.aaai.org/Papers/AIIDE/2008/AIIDE08-039.pdf.

[53] D. Thue, V. Bulitko, M. Spetch, and E. Wasylishen. Learning Player Preferences to Inform Delayed Authoring. Papers from the AAAI Fall Symposium on Intelligent Narrative Technologies. FS-07-05: pp. 158-161. AAAI Press. Arlington, Virginia, USA. November 9, 2007. http://ircl.cs.ualberta.ca/files/webfm/games/pubs/thue07-aaaisymp.pdf.

[54] The Typing of the Dead. Smilebit. http://www.neoseeker.com/Games/Products/DC/typingdead/.

[55] Unreal Tournament. Epic Games and Digital Extremes. GT Interactive. http://www.unreal.com/.

[56] Vampire: The Masquerade – Bloodlines. Troika Games. Activision.

[57] Warcraft III: Reign of Chaos. Blizzard Entertainment. Vivendi. http://www.blizzard.com/us/war3/.

[58] World of Warcraft. Blizzard Entertainment. Vivendi. http://www.worldofwarcraft.com/index.xml.

[59] M. Yi. They Got Game. San Francisco Chronicle. December 18th, 2004. http://www.sfgate.com/cgi-bin/article.cgi?f=/chronicle/archive/2004/12/18/MNGUOAE36I1.DTL.

[60] S. Zeitchik. 'Deathly Hallows' sells 8.3 million. Variety. July 22nd 2007. variety.com/article/VR1117968991.html?categoryid=21&cs=1.

Appendices

Appendix A - Quest Catalogue

Each pattern’s description is from the quest catalogue.

Quest Points

- **Acquire**: This quest point succeeds when the PC acquires a specified number of items of the specified blueprint.
- **Alarm**: This quest point succeeds at a specific hour of the day, regardless of the activities of the PC. If time is artificially forwarded past this hour, it will succeed immediately after time is forwarded. Note that the hour of the day will not be checked until the quest point is enabled.
- **Approach**: This quest point succeeds when the PC approaches within a specified distance of the target and this quest point is enabled. Optionally, a caption can appear above the PC. If no caption is desired then leave the 'Caption' field empty or remove the 'Module heartbeat - speak caption' encounter.
- **Area barrier falls**: This quest point will automatically succeed and drop an area barrier created by the 'Area barrier rises' quest point. When the PC clicks on the 'Area Transition' they will be transported to the destination and 'Travel Time' hours will elapse.
- **Area barrier rises**: This quest point will automatically succeed. When it does, it will turn an 'Area Transition' into a barrier. Whenever the 'Area Transition' is clicked the PC will begin an 'Introspection' conversation. The barrier can be dropped with an 'Area barrier falls' quest point.
- **Arrive**: This quest point succeeds when the PC enters the area and this quest point is enabled. Optionally, a caption can appear above the PC. If no caption is desired then leave the 'Caption' field empty or remove the 'Area enter - speak caption' encounter.
- **Arson**: This quest point succeeds if the PC burns a placeable. The PC can burn the placeable by possessing an item used to ignite placeable and then selecting a specified dialogue line while speaking to the placeable. The burning will last a specified length of time after which the 'Original placeable' will be replaced by the 'Replacement placeable'. The quest point will fail if the original placeable is destroyed before it can be burned.
- **Automatic**: This quest point will succeed as soon as it is enabled.

- **Converse**: This quest point succeeds if the 'Success Line' of dialogue is reached and fails if the NPC who speaks the 'Success Line' is killed.
- **Converse – exchange items**: This quest point succeeds when a specified line of dialogue is reached. Upon being reached, specified items are exchanged between the PC and the NPC. The quest point fails if the NPC involved in the exchange dies.
- **Converse – give items**: This quest point succeeds when a specified line of dialogue is reached. Upon being reached, the specified items are transferred. The quest point fails if the NPC involved in the transfer dies.
- **Converse – has items**: This quest point succeeds when a specified line of dialogue is reached and a character (PC or NPC) is holding a specified number of a specified item. This quest point fails if the NPC dies.
- **Converse – use skill**: This quest point succeeds when the PC successfully uses a skill in a conversation with an NPC. If the skill check fails, this quest point will fail. The quest point will also fail if the NPC dies.
- **Earn**: This quest point succeeds when the PC sells good worth a specified amount of money to a specified merchant. The money can be accumulated over several transactions. This quest point will fail if the merchant dies.
- **Equip**: This quest point succeeds when the player equips a specified item.
- **Fail**: This quest point succeeds as soon as it is enabled, making it functionally identical to an ‘Automatic’ quest point. This quest point does not cause the quest to fail, but can be used to explicit mark that a quest will fail if a committing dead end branch with this quest point succeeds.
- **Filter amount**: This quest point succeeds when there is the right amount of a (usually intangible) quantity. For example, ensuring the PC has enough fame or that the PC is in jail could be handled with this quest point. The 'Set amount' action must be used to change the desired amount relevant to this quest point. The 'Set amount' action does not need to be contained in this quest point. To clarify, if during the progress of one quest the player's fame increases then the 'Filter amount' quest point set to fame in another quest can succeed.
- **Filter class**: This quest point succeeds if the PC has at least 1 level in the given class. This quest point is meant to serve as a guard for another quest point with class-specific content. For example, if a rogue 'Filter Class' quest point is placed before a 'Converse' quest point with a conversation tailored for rogues then the 'Converse' quest point can only be reached if the PC has a level in the rogue class. If the PC acquires another class, the class requirement will be checked again using the classes of the PC.

- **Filter gender**: This quest point succeeds if the PC is the given gender. This quest point is meant to serve as a guard for another quest point with gender-specific content. For example, if a male 'Filter gender' and female 'Filter gender' quest points each guarded a different 'Converse - give items' quest point then it would allow a different item to be given the PC depending on whether the PC is male or female.
- **Filter level**: This quest point succeeds if the PC has at least the given number of levels. This quest point is meant to serve as a guard for another quest point with level-specific content, such as preventing a weak PC from acquiring a quest that is too difficult. If the PC acquires more levels, the level requirement will be checked again using the new PC level.
- **Filter race**: This quest point succeeds if the PC is a given race. This quest point is meant to serve as a guard for another quest point with class-specific content. For example, if a dwarf 'Filter Race' quest point is placed before a 'Converse' quest point with a conversation tailored for dwarves then the 'Converse' quest point can only be reached if the PC is a dwarf.
- **Join**: This quest point succeeds when it is enabled by a set number of quest points. This is a specialization of the 'Automatic' quest point and is intended for use when multiple branches of a quest join back together.
- **Kill**: This quest point succeeds if a specified number of NPCs of a specified blueprint are killed.
- **Leave area**: This quest point succeeds when a specified NPC leaves the area. The quest point fails in the NPC dies.
- **Menu choice**: This quest point succeeds when the PC chooses a specfied option in the menu.
- **Minigame**: This quest point succeeds when the PC wins a specified minigame.
- **Murder**: This quest point will succeed when the PC kills a good-aligned creature of a specific blueprint.
- **NPC perceives NPC**: This quest point will succeed when an NPC of a specified blueprint perceives a NPC of another specified blueprint. The quest point will fail if either the observing NPC or the target NPC dies.
- **NPC perceives PC**: This quest point will succeed when an NPC of a specified blueprint perceives the PC. The quest point will fail if the observing NPC dies.
- **Open placeable**: This quest point succeeds when a placeable is opened. The quest point will fail if the placeable is destroyed.
- **Party member**: This quest point succeeds when a specified NPC joins the PC's party.

- **Pay**: This quest point succeeds when the 'Pay Line' is reached and the PC pays an amount of gold to an NPC. If the PC does not have enough gold, the 'Pay Line' will not be displayed. The quest point fails if either the 'No Pay Line' is reached or if the NPC dies.
- **PC dies**: This quest point succeeds when the PC dies.
- **Place item**: This quest point succeeds when the specified number of items of a specified blueprint is placed into the placeable's inventory. The quest point fails if the specified placeable is destroyed.
- **Quest completed**: This quest point will be reached when another specified quest is completed.
- **Rest**: This quest point succeeds when the PC rests. The quest point must be enabled before it can be reached.
- **Status effect**: This quest point will succeed when a creature has a specified status effect, such as blinded, charmed, or stunned. The status effect can be delivered by spell or by physical attacks. The quest point will fail if the creature is killed.
- **Steal**: This quest point will succeed when the PC acquires an item belonging to a NPC. Specifying that an item belongs to a NPC is done using the 'Set item owner' action. Regardless of how the item is acquired – such as picking it off the ground, through conversation, or pick pocketing – if the item is marked as belonging to an NPC then it is stolen. The items can be disowned, so they can legally acquired by the PC, by the 'Disown items' action.
- **Timer aborts**: This quest point will automatically succeed and abort a specified timer. The corresponding 'Timer expires' quest point will never fire. If the timer is aborted before the 'Timer starts' quest point is reached then the 'Timer starts' quest point can still succeed but the timer will not be activated. An aborted timer cannot be reactivated.
- **Timer expires**: This quest point will succeed when a specified timer expires. That timer must have first been started with a 'Timer starts' quest point and cannot have been aborted with a 'Timer aborts' quest point. The rate at which game time passes is adjusted in the Aurora Toolset. This quest point will also print out periodic warning messages consisting of the 'Warning Prefix', the remaining seconds, and the 'Warning Suffix'. If no warning message is desired leave both the 'Warning Prefix' and 'Warning Suffix' empty.
- **Timer starts**: This quest point succeeds automatically and starts the specified timer. That timer can be aborted with a 'Timer aborts' quest point. When the timer expires the corresponding 'Timer expires' quest

point will succeed. The rate at which game time passes is adjusted in the Aurora Toolset.

- **Trigger**: This quest point succeeds when the PC enters a trigger and this quest point is enabled. Optionally, a caption can appear above the PC. If no caption is desired then leave the 'Caption' field empty or remove the 'Trigger enter - speak caption' encounter.
- **Unequip**: This quest point succeeds when the player unequips an item.
- **Use item**: This quest point succeeds when the PC uses a specified item.
- **Use item with placeable**: The quest point will succeed if the PC uses a specified placeable with a specified item in their inventory. The item is consumed when the placeable is used. This quest point will fail if the placeable is destroyed.
- **Use placeable**: This quest point will succeed when the PC uses a specified placeable. This quest point will fail if the placeable is destroyed.
- **Use spell on NPC**: This quest point succeeds if a specified spell is cast on a specified NPC. The quest point will fail if the NPC dies.
- **Use spell on placeable**: This quest point will succeed when a PC uses a specified spell on a specified placeable.
- **Wound**: This quest point will succeed if the specified creature is wounded. Optionally, if the damage type matters then the quest point will only succeed if the creature is wounded by the specified type of damage. This quest point will fail if the creature dies before being wounded.

Meta Quest Points

- **Activate placeable**: This meta quest point represents how the the PC activates a placeable. There are several recommended bindings for this meta quest point:
  - Open placeable – The PC opens the placeable to activate it
  - Use placeable – The PC activates placeable item by using it normally
  - Use item with placeable – The PC activates an item by consuming an item in their inventory
  - Use spell on placeable – The PC activates the placeable by casting a spell on it
- **Arrange meeting**: This meta quest point represents how a meeting is arranged with the PC. There are several recommended bindings for this meta quest point:
  - Acquire – The PC obtains an item describing the meeting place

  - Automatic – The meeting does not need to be arranged
  - Converse – The meeting is arranged during a conversation

- **Discover**: This meta quest point represents how a person or object is found after a search. There are several recommended bindings for this meta quest point:
  - Alarm – The search is over at a certain time of day
  - Approach – The search is over when the PC approaches the target
  - Arrive – The search ends when the PC arrives in an area
  - Converse – The search is over when the PC talks to an NPC about the target and gets a definitive answer
  - Trigger – The search is over when the PC enters a region in an area that may or may not contain the target
  - Use placeable – The search ends when an placeable is used

- **End**: This meta quest point represents how a quest ends and is intended to be used as the last quest point of a quest. There are several recommended bindings for this meta quest point:
  - Arrive – The quest end when the PC enters an area
  - Automatic – The quest ends automatically after the previous quest point (presumably the climax) is reached
  - Converse – The quest ends when the PC talks to an NPC
  - Converse - exchange items – The quest ends when the PC and an NPC exchange items
  - Converse - give items – The quest ends when the PC gives the NPC an item or vice versa
  - Kill - The quest ends when a NPC, such as the villain, dies
  - Place item – The quest ends when a PC places an item in a container
  - Use placeable – The quest ends when a placeable, such as an escape pod, is used

- **Escort begins**: This meta quest point represents how an escort begins. Either the PC or an NPC can be the escorter. There are several recommended bindings for this meta quest point:
  - Approach – The escort begins with the PC approaching an NPC
  - Arrive – The escort begins when the PC enters an area
  - Converse – The escort begins with the PC talking to an NPC
  - Converse - give items – The escort begins when items are transferred through conversation
  - Kill – The escort begins when an NPC, who was presumably guarding the follower, dies
  - NPC perceives PC – The escort begins when the follower perceives the PC
  - Pay – The escort begins with the PC paying an NPC

  - Trigger – The escort begins with the PC entering a region within the area.

- **Escort ends**: This meta quest point represents how an escort ends. Either the PC or an NPC can be the escorter. There are several recommended bindings for this meta quest point:
  - Approach – The escort ends with the PC approaching an NPC
  - Arrive – The escort ends when the PC enters an area
  - Converse – The escort ends with the PC talking to an NPC
  - Converse - give items – The escort ends when items are transferred through conversation
  - Kill – The escort ends when an NPC dies
  - NPC perceives NPC – The escort ends when the follower perceives a different NPC
  - Trigger – The escort ends with the PC entering a region within the area.

- **Gain authority**: his meta quest point determines how a PC gains authority over an NPC. There are several recommended bindings for this meta quest point:
  - Acquire – The PC acquires items to gain authority
  - Converse – The PC is deputized through conversation with an authority
  - Converse - exchange items – The PC gains authority by exchanging items with the NPC
  - Converse - give items – A transfer of items between the PC and the NPC gives authority
  - Converse - has item – The PC gains authority by showing an NPC an item during conversation
  - Converse - use skill – The PC gains authority through skill in conversation
  - Convince – The PC gains authority by convincing an NPC in one of a variety of ways
  - Filter class – The PC gains authority because of their profession
  - Filter gender – The PC gains authority because of their gender
  - Filter level – The PC gains authority because of their previous experience
  - Filter race – The PC gains authority because of their race
  - Pay – The PC gains authority through bribery
  - Retrieve – The PC gains authority by retrieving items for an authority

- **Gain item**: This meta quest point represents how the PC gains an item. There are several recommended bindings for this meta quest point:
  - Acquire – The PC gains the item by any means, including picking it off the ground

  - Converse - give items – The PC gains the item through a conversation

- **Give item**: This meta quest point represents how the PC gives an item to an NPC or organization. There are several recommended bindings for this meta quest point:
  - Converse - exchange items – The PC and an NPC exchange items
  - Converse - give items – The PC gives the item to an NPC during a conversation
  - Converse - has item – The PC merely has to show the item during a conversation
  - Place item – The PC gives the item by putting it in a placeable
  - Use item with placeable – The PC sacrifices the item to use a placeable.
- **Hint**: This meta quest point represents how the PC receives a hint on how to complete the current quest. There are several recommended bindings for this meta quest point:
  - Acquire – The PC receives a hint by acquiring an item
  - Automatic – There are no hints to receive
  - Converse – The PC receives a hint by talking to an NPC
  - Converse - exchange items – The PC receives a hint through an exchange of items
  - Converse - give items – The PC receives a hint by giving or receiving an item
  - Converse - has item – An NPC will give the PC a hint if the PC possesses an item
  - Convince – The PC receives a hint by convincing an NPC
- **Hire**: This meta quest point represents how the PC is hired by another character. There are several recommended bindings for this meta quest point:
  - Converse – The PC is hired after a conversation with an NPC
  - Converse - give items – The PC is hired after a conversation with an NPC where one of them receives an item
- **Negotiate**: This meta quest point represents the PC conversing with an NPC. There are several recommended bindings for this meta quest point:
  - Converse – The PC simply talks to the NPC
  - Converse - exchange items – The PC exchanges items with the NPC during conversation
  - Converse - give items – The PC gives/receives items to/from the NPC during conversation
  - Converse - has item – The PC can only talk to the NPC while possessing an item

  - Converse - use skill – The PC can only talk to an NPC after manipulating them
  - Pay – The PC can only talk to an NPC after paying them

- **Obtain proof**: This meta quest point represents whether proof of a deed needs to be acquired and delivered. There are several recommended bindings for this meta quest point:
  - Automatic – There is no need to acquire nor deliver proof
  - Retrieve – The PC must acquire the proof and deliver it

- **Start**: This meta quest point represents how a quest begins and is intended to be used as the first quest point of a quest. There are several recommended bindings for this meta quest point:
  - Acquire – The quest begins when the PC acquires an item
  - Approach – The quest begins when the PC approaches something
  - Arrive – The quest begins when the PC arrives in an area
  - Automatic – The quest begins as soon as the module is loaded
  - Converse – The quest begins through conversation
  - Converse - give items – The quest begins when an item is transferred between the PC and an NPC during conversation
  - Converse - use skill – The quest begins through the PC's skill in conversation
  - Equip – The quest begins when the PC equips an item
  - Filter class – The quest begins if the PC has a level in a specified class
  - Filter gender – The quest begins if the PC is a specified gender
  - Filter level – The quest begins if the PC has enough experience
  - Filter race – The quest begins if the PC is the specified race
  - Kill – The quest begins when the PC kills a specific NPC
  - Quest completed – The quest begins when another quest is completed
  - Pay – The quest begins when the PC pays an NPC
  - Trigger – The quest begins when the PC enters a region of an area

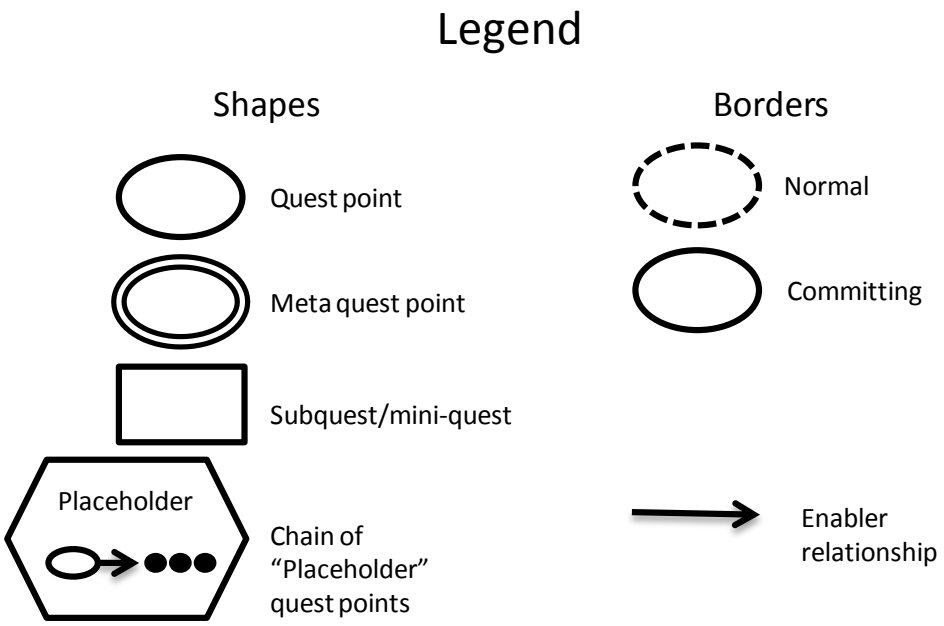
- **Start ambush**: This meta quest point represents what causes an ambush to be sprung. There are several recommended bindings for this meta quest point:
  - Approach – The ambush begins when the PC approaches an NPC
  - Arrive – The ambush begins when the PC enters an area
  - Converse – The ambush begins when the PC talks to an NPC
  - Trigger – The ambush begins when the PC enters a region inside of an area

- **Track**: This meta quest point represents how the PC tracks down a person or object. There are several recommended bindings for this meta quest point:

- Acquire – The PC acquires an item which gives a clue about the target's location
- Arrive – The PC gains a clue by arriving in an area
- Automatic – There is no trail of clues and the track succeeds automatically
- Converse – The PC gains a clue by talking to an NPC
- Converse - give items – Through conversation the PC acquires an item with a clue
- Converse - use skill – The PC receives a clue through skill in conversation
- Convince – The PC convinces an NPC to give a clue
- Journey – The PC gains a clue by completing a journey
- Open placeable – The PC receives a clue by looking into a container
- Trigger – The PC gains a clue by entering a region of an area
- Wait – The PC gains a clue by waiting for someone or something

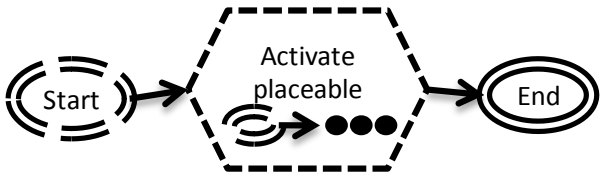
- **Travel**: This meta quest point represents how the game determines the PC has reached a location. There are several recommended bindings for this meta quest point:
  - Approach – The PC reaches a location by approaching an NPC
  - Arrive – The PC reaches a location by entering an area
  - Trigger – The PC reaches a location by entering a region within an area

Quest Patterns



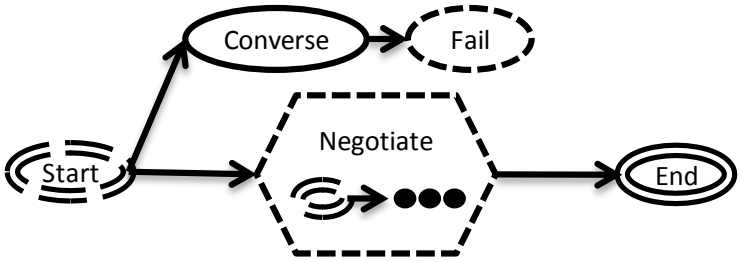
Activate

The PC must activate several placeables. The 'Activate' meta quest point describes how a single placeable is activated. The 'Activate' meta quest point can be chained to create a series of placeables to be activated.



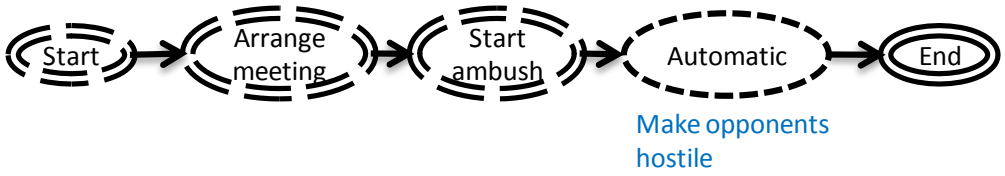
Advocate

The PC must converse with a chain of NPCs. Each link of the chain is a 'Negotiate' meta quest point that requires the PC to reach a line of dialogue in a conversation. There is only one 'Negotiate' meta quest point initially and more may be created through chaining. If the PC reaches the failure dialogue line then the quest immediately fails.



Ambush

There will be an ambush with either the PC or an NPC as the target. First, a meeting must be arranged between the opponent and the PC. What initiates the ambush is determined by a 'Start ambush' meta quest point. Finally, the opponent and PC become hostile to each other. The opponent will attack the PC. Any NPCs which belong to the same faction as the opponent will also become hostile but will not attack the PC until the PC attacks a member of their faction. The author may order other faction members to instantly attack the PC on a member-by-member basis.

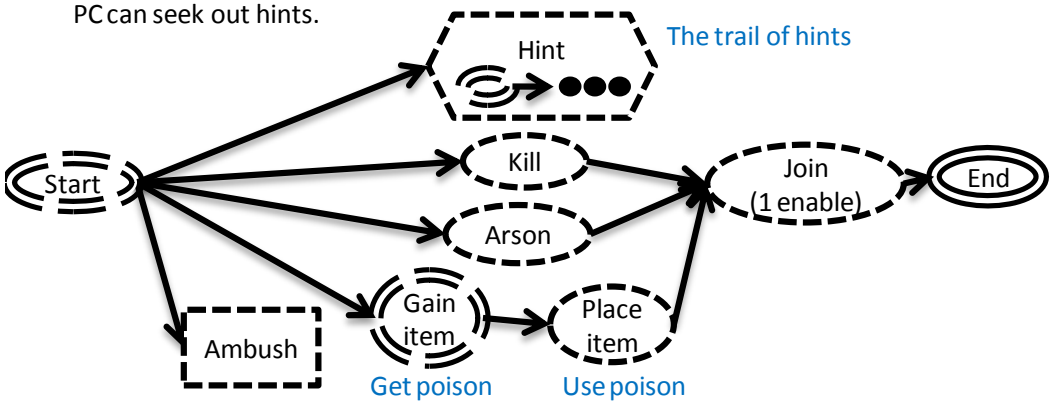


Area barrier

The PC is prevented from using an area transition until the contained mini-quest is completed. Whenever the PC attempts to use the specified area transition, an introspection conversation will begin instead. What mini-quest the PC must do between the area barrier rising and falling is initially represented by a 'Placeholder' mini-quest which should be replaced. When that mini-quest is completed the area barrier will be removed automatically.

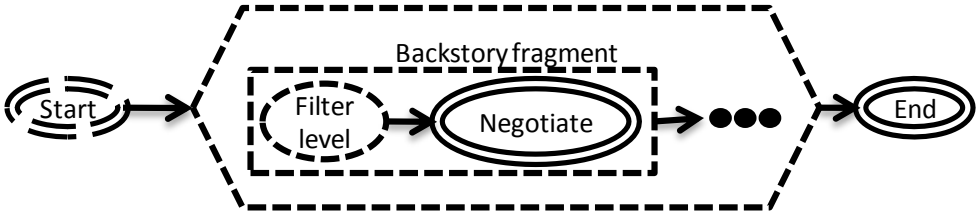
Assassinate

The PC must assassinate an NPC. There are a variety of ways the victim can be assassinated: directly by killing the victim, indirectly by committing arson, or indirectly by poisoning. Optionally, the victim can be lured into an ambush or the PC can seek out hints.



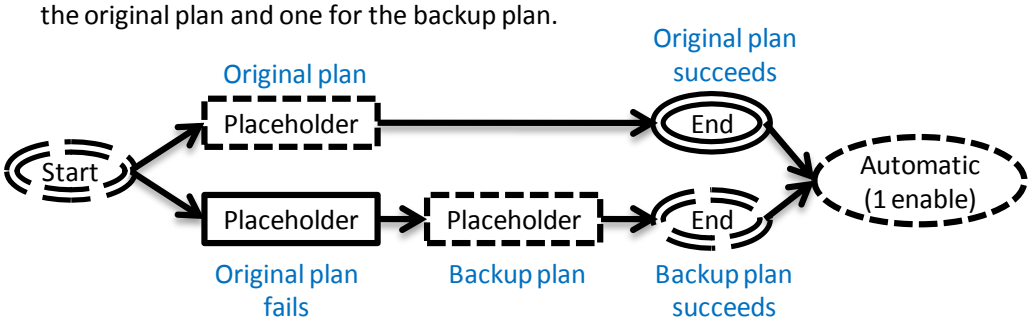
Backstory

The PC learns a little about an NPC's backstory whenever the PC reaches specific levels. Each 'Backstory fragment' mini-quest contains a 'Filter level' quest point to provide a level threshold and a 'Negotiate' meta quest point to determine how the PC learns the backstory fragment. Initially, there is only a single 'Backstory fragment' but more can be created through chaining that mini-quest.



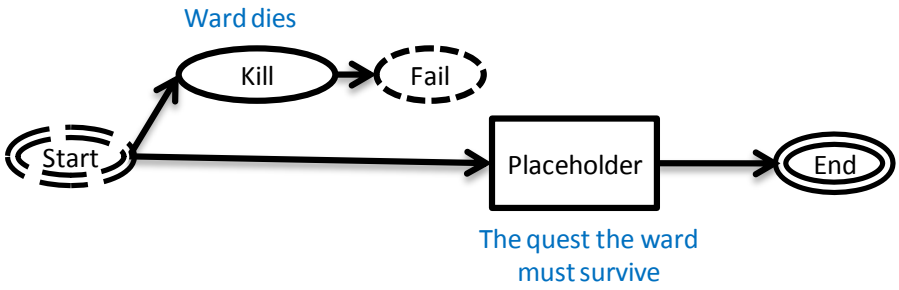
Backup plan

The PC attempts to complete the original plan (which should replace the 'Placeholder' subquest). If the 'Original plan fails' quest point succeeds then the PC must complete a backup plan (also initially a 'Placeholder'). There are two separate successful endings corresponding to two 'End' meta quest points: one for the original plan and one for the backup plan.



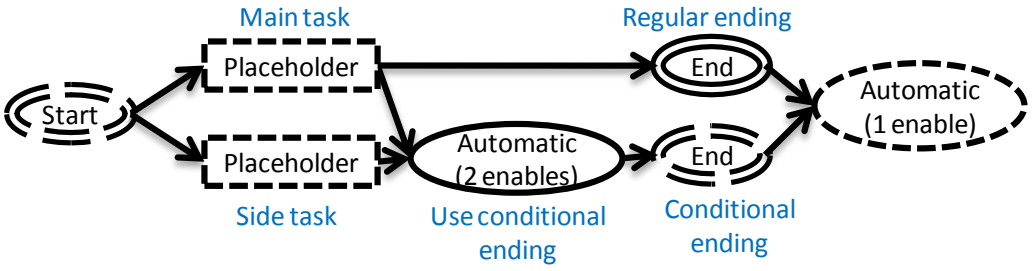
Bodyguard

The PC must protect a specified NPC until another task (represented by the 'Placeholder' miniquest) is completed. If that NPC dies before that task is completed, the quest fails. Otherwise, the quest succeeds.



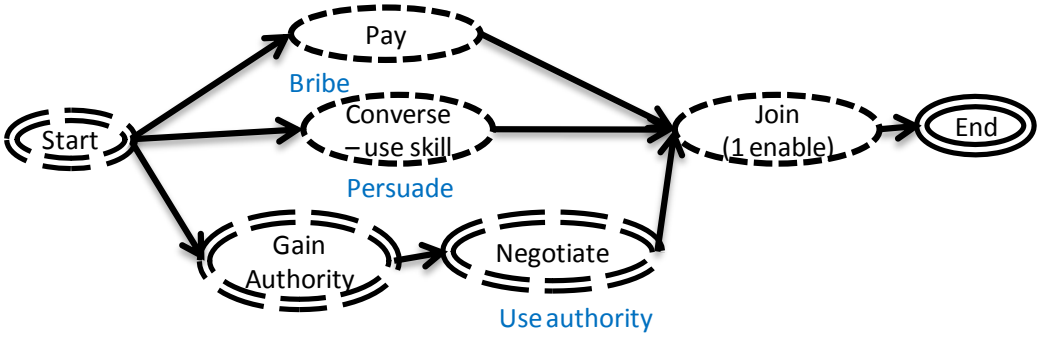
Conditional reward

The PC must perform some task represented by the main subquest. If the PC completes an optional bonus/penalty subquest they will receive a different reward provided by different 'End' meta quest points.



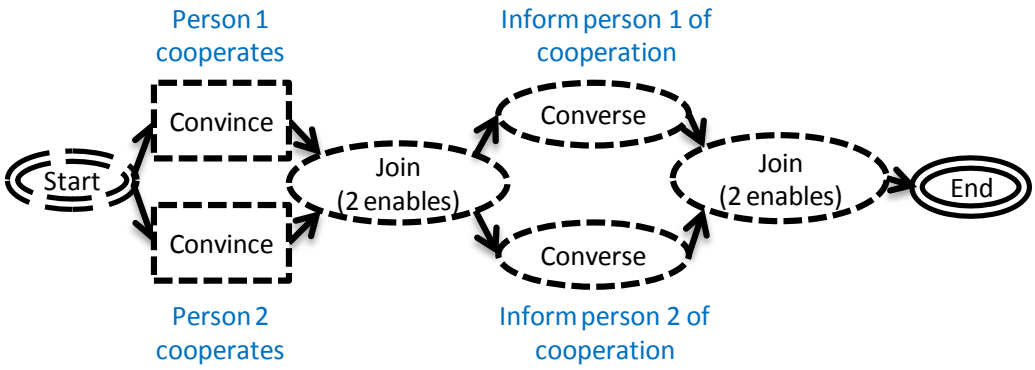
Convince

The PC must convince another NPC on an issue. There are many way to convince an NPC and only one way needs to be successful: bribery, conversational skill, and use of another's authority. This quest will fail if the NPC is killed or every method of convincing the NPC fails.



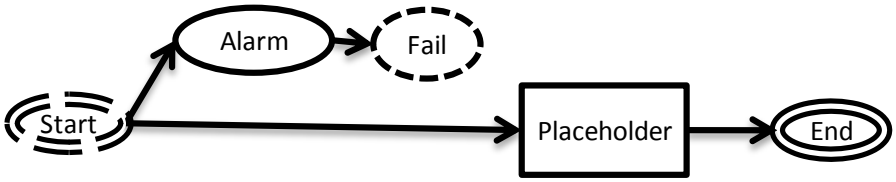
Cooperate

The PC must convince 2 people to cooperate. It does not matter in which order the people are convinced. After both people have been convinced the PC must then inform both people, again in any order, that an agreement has been reached.



Deadline

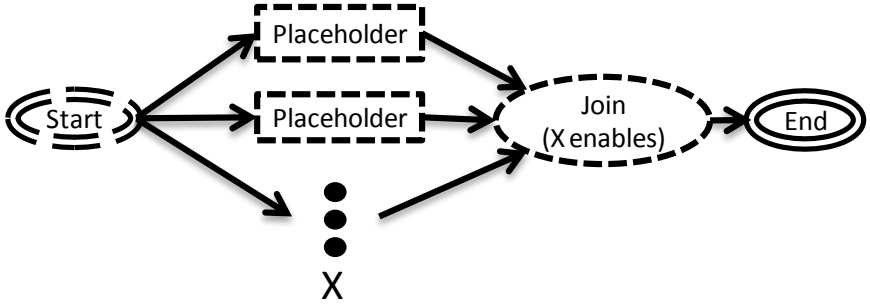
The PC must perform a task (which should replace the 'Placeholder' subquest) before the deadline or else the quest will fail. The deadline is an hour of the day, so the task should take less than a day to complete.



The quest under
a time limit

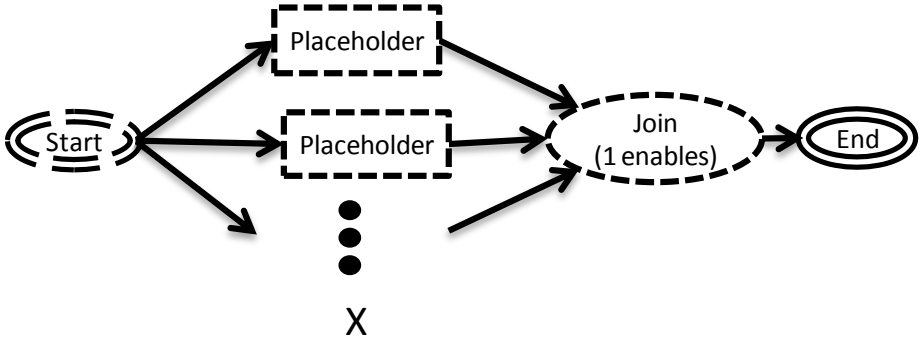
Do in any order

The PC must do all of the the subquests to complete this quest, however the subquests may be done in any order. The default pattern has 2 subquest branches but it is easy to create more branches. Each branch must be enabled by the 'Start' meta quest point and the 'Join' quest point must be enabled by that branch. The 'Minimum # of enablers' in the 'Join' quest point should be equal to the number of branches.



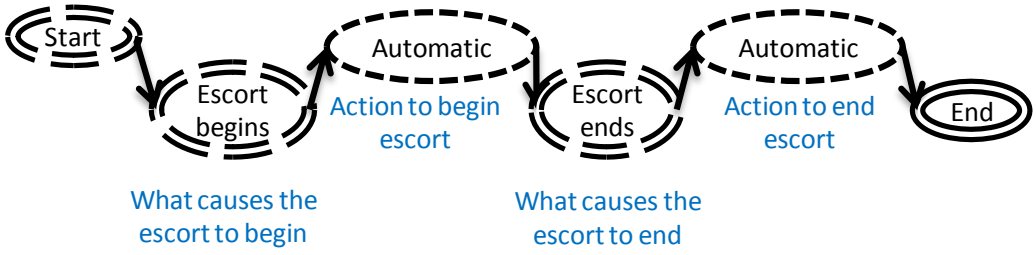
Do one of many

The PC needs to only complete 1 of the subquests to complete this quest. The default pattern has 2 subquest branches, but it is easy to create more branches. Each branch must be enabled by the 'Start' meta quest point and the 'Join' quest point must be enabled by that branch.



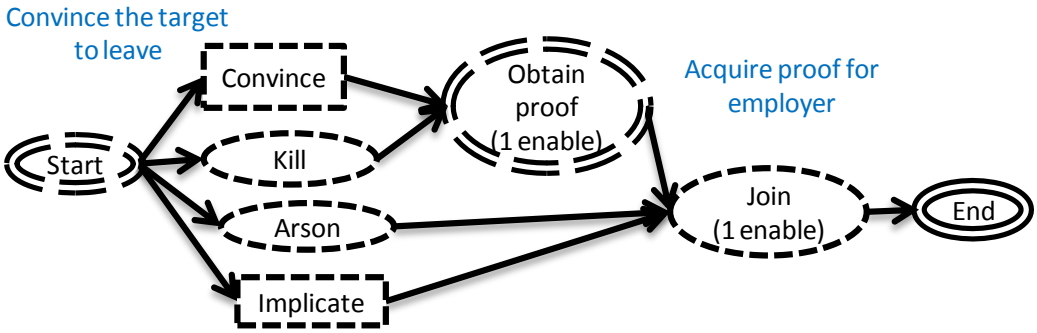
Escort

The PC must escort an NPC. What causes the escort to start is determined by the 'Escort begins' meta quest point. During the escort the NPC will follow the NPC. What causes the escort to end is determined by the 'Escort ends' meta quest point.



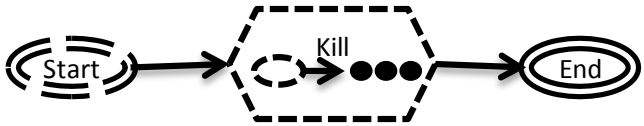
Expel

The PC must expel an NPC victim from a location. There are 4 ways of expelling the victim: 1) Convince them to leave 2) Kill them 3) Burn their property 4) Implicate them. Only 1 of these 4 methods needs to succeed to complete this quest. If the PC convinces the victim to leave or kills the victim then proof may need to be delivered to the employer.



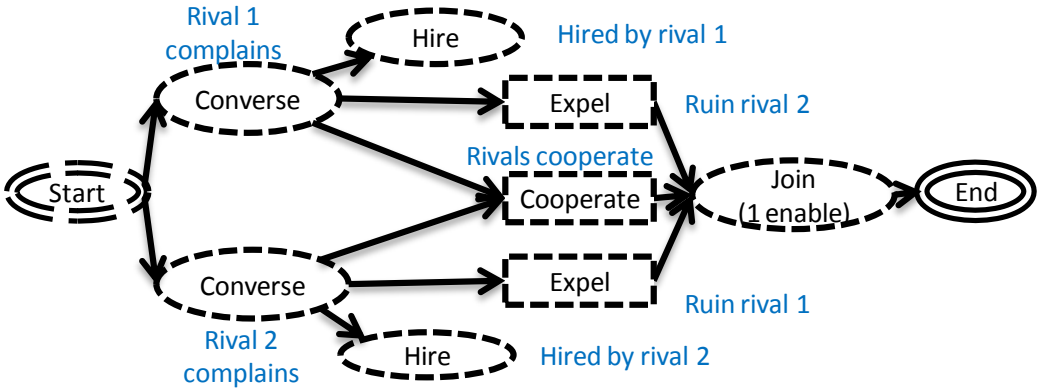
Exterminate

The PC must kill at least one group of NPCs. Each group is represented by a 'Kill' quest point that requires the PC to kill a specified number of creatures of a specified blueprint. The 'Kill' quest points can be chained if more than one group of creatures must be killed.



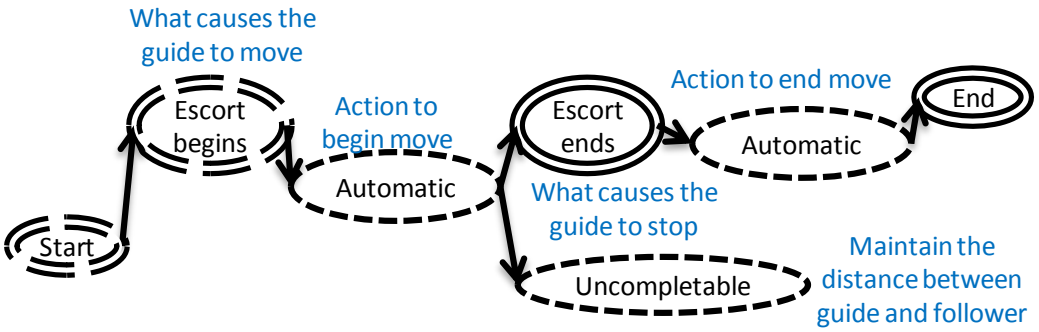
Feud

The PC must resolve a feud between two NPCs or their respective organizations. Each of the rivals can hire the PC to expel the other rival. The quest can end in one of three ways: The first rival is expelled, the second rival is expelled, or both rivals are convinced to cooperate.



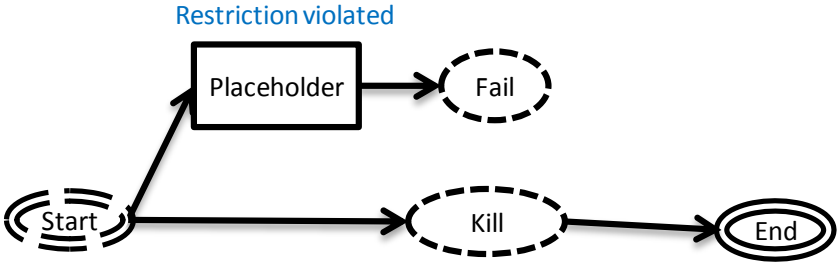
Follow

The PC must follow an NPC from one place to another. How the following begins and ends is represented with the 'Escort begins' and 'Escort ends' meta quest points.



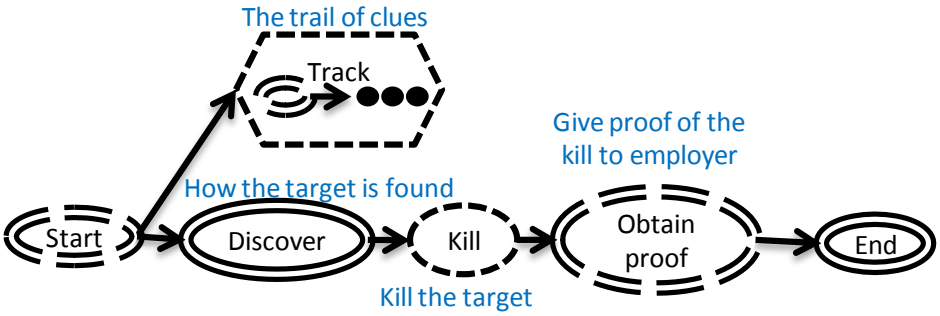
Gladiator

The PC must kill the target while obeying some restriction. That restriction should replace the 'Placeholder' subquest. The quest will fail if the PC ever violates that restriction. If no restriction is desired then leave its placeholder alone.



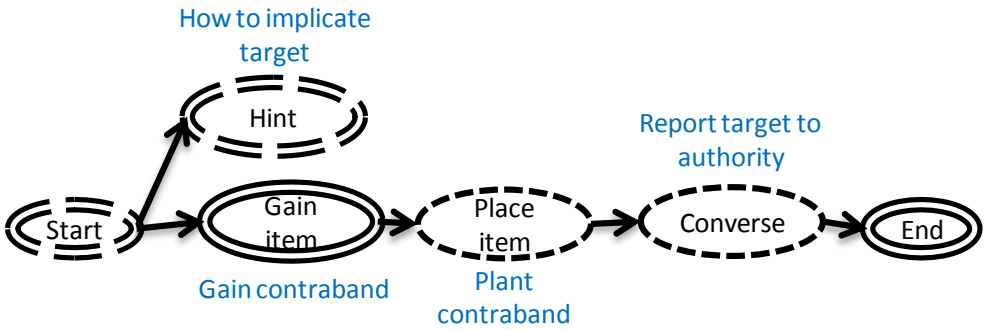
Hunt

The PC must hunt a group of creatures and kill them. There is an optional chain of 'Track' meta quest points to provide a trail of clues. Once the creatures have been killed the 'Obtain proof' meta quest point can require the PC to deliver items, such acquiring kill trophies from the creatures and delivering them to the employer.



Implicate

The PC must implicate an NPC by planting an item in a container and reporting the possession to an NPC authority figure. There is an optional meta quest point that can be used to provide the PC with a hint, such as to the location of the implicating item.

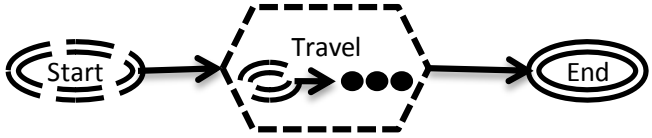


Investigate

The PC must converse with a chain of NPCs. Each link of the chain is a 'Negotiate' meta quest point that requires the PC to reach a line of dialogue in a conversation. There is only one 'Negotiate' meta quest point initially and more may be created through chaining.

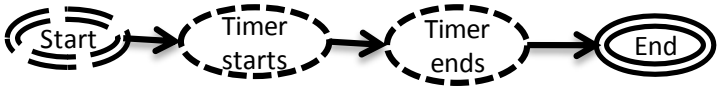
Journey

The PC must travel somewhere. The travel meta quest point describes how one stage of the journey is reached. The 'Travel' meta quest point may be chained to create a journey with multiple stages.



Pause

This quest will succeed after a specified number of seconds have elapsed. No action by the PC is required or expected. This quest should be used as a mini-quest to create a pause between the events of another quest. The ‘Timer name’ option of each ‘Pause’ quest must have a unique name.

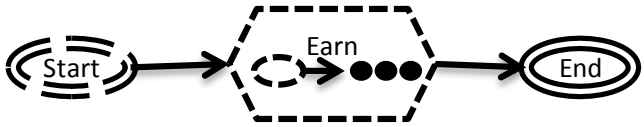


Placeholder

This is a placeholder quest. It should be replaced with another quest pattern.

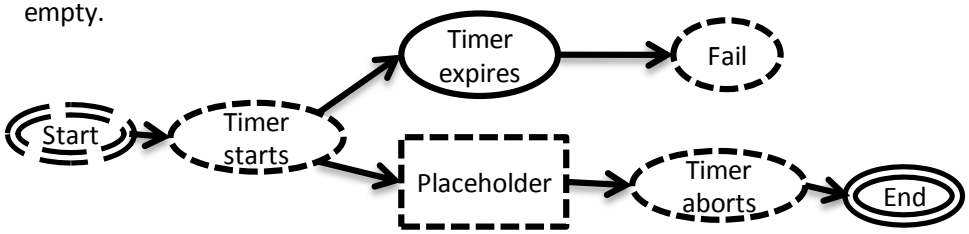
Profit

The PC must earn a specified amount of money by selling goods to a specified merchant. Chaining the 'Earn' quest point and increasing their 'Money' option allows the game to respond to the increasing wealth of the PC.



Race against time

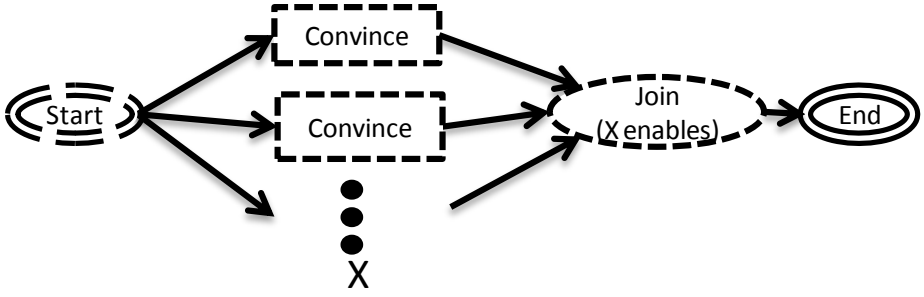
The PC must perform a task under a time limit or else the quest will fail. The task that must be completed is represented with a placeholder quest, which should be replaced with the appropriate quest or quest point. While the countdown timer is active, a warning will periodically appear above the PC's head. If no warning message is desired leave both the 'Warning prefix' and 'Warning suffix' empty.



The quest under a time limit

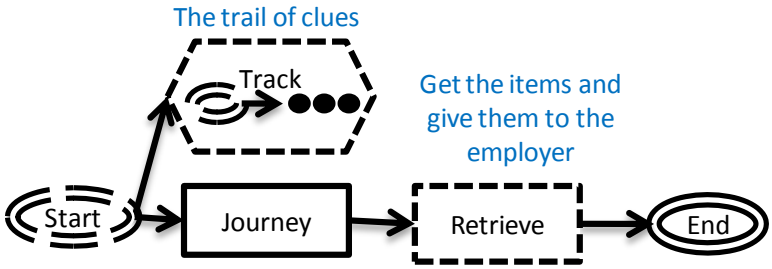
Rally

The PC must convince several people independently from each other. For example, the PC must convince each lord on a council to elect the PC as emperor. The default pattern has 2 subquest branches but it is easy to create move branches. Each branch must be enabled by the 'Start' meta quest point and the 'Join' quest point must be enabled by that branch. The 'Minimum # of enablers' in the 'Join' quest point should be equal to the number of branches.



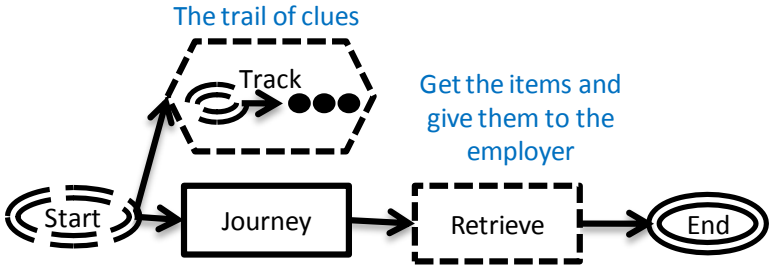
Recovery Expedition

The PC must journey to acquire a set of items and deliver them. Optionally, the PC may receive a series of clues by using a chain of 'Track' meta quest points.



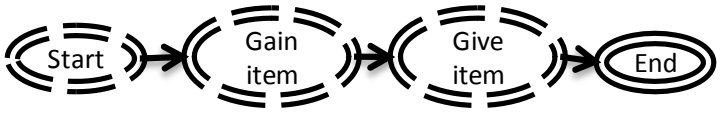
Recovery Expedition

The PC must journey to acquire a set of items and deliver them. Optionally, the PC may receive a series of clues by using a chain of 'Track' meta quest points.



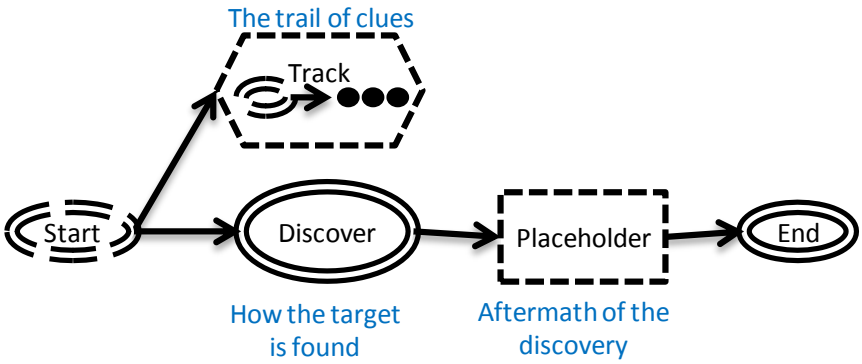
Retrieve

The PC must acquire items and deliver them. How they are to be acquired and delivered is represented with the 'Gain item' and 'Give item' meta quest points respectively.



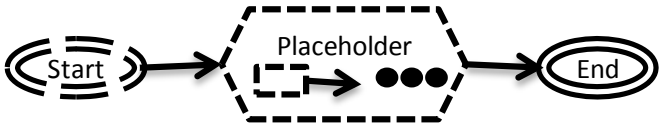
Search

The PC must find a person, place, or thing. There is an optional chain of 'Track' meta quest points that provide clues to lead the PC to the target. The 'Discover' meta quest point specifies how the target is found. The placeholder subquest should be replaced with whatever should happen when the target is found.



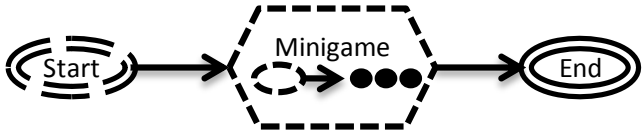
Subquest chain

The PC must complete a chain of subquests. Initially, there is only 2 subquests in the chain but more can be easily added through the insert or chain operation.



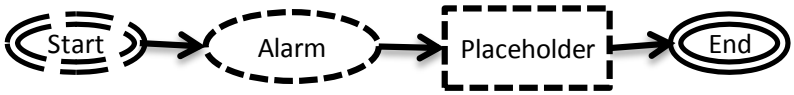
Tournament

The PC must win a chain of minigames. The PC will fail the quest if they lose any of the minigames. There is only one 'Minigame' quest point initially and more can be created through chaining.



Wait

The PC must wait for a specific hour of the day. What the PC must do next when that hour arrives should replace the ‘Placeholder’ subquest. An example of a 'Wait' quest would be waiting until midnight for an informant to appear in an alley and then talking to the informant to gain vital information.



Appendix B - Participant Briefing

You are invited to participate in a study being conducted by Marcus Trenton, under the supervision of Dr Duane Szafron. The study examines the effectiveness of two different tools, the Aurora Toolset's NWScript and ScriptEase, in creating scripts for quests in the computer game Neverwinter Nights. Due to the technical nature and lengthy duration of scripting, only those with previous experience with a scripting or programming language and an interest in computer games qualify for this study. Tutorials with examples for both the Aurora Toolset's NWScript and ScriptEase will be provided for reference.

The study is scheduled for either Saturday, January 17th or Saturday, January 24th in the Games and Graphics Lab, CSC 105. The study will occur from 10am to 5pm with an hour lunch break where drinks and pizza will be provided. Since this study is lengthy, participants will be paid fifty dollars compensation for participation.

If you are interested in or would like to know more then please reply to this email to reserve one of the twenty seats in the study, and please indicate which of the dates you would be available.

Appendix C - Consent Form

The original formatting has been altered to fit the dissertation format.

Research Information and Participants' Consent Form: Is ScriptEase More Effective at Scripting than the NWScript?

Purpose. You are invited to participate in a research study *Is ScriptEase More Effective at Scripting than the NWScript?* being conducted by Dr. Duane Szafron and Marcus Trenton of the Department of Computer Science, University of Alberta. The study examines the effectiveness of two different tools, the Aurora Toolset and ScriptEase, in creating scripts for the computer game Neverwinter Nights. We are interested in which tool is easier to use, results in fewer errors, and is faster to use.

Your participation. Your participation involves first filling out a pre-experiment demographics survey. Then you will use the Aurora Toolset and ScriptEase tools to script up to eight quests per tool for the game Neverwinter Nights and use that game to test them. Finally, you will complete a questionnaire about ease of use, errors encountered, speed of use, and overall preferences for each tool. The experiment should take about six hours to complete. You will receive $50 compensation for participation.

Your rights. Your decision to participate in this study is entirely voluntary and you may decide at any time to withdraw from the study. Your decision to discontinue will not affect your academic status or access to services from the University of Alberta. If you choose to participate, you may skip any items you do not wish to answer. Responses made by individual participants on the questionnaire will remain confidential, and your name will not appear on the questionnaire or be associated with your responses in any way. Questionnaires/scripts will be identified only by a researcher-assigned code number. Only researchers associated with the project will have access to the questionnaires. The results of this study may be presented at scholarly conferences, published in professional journals, or presented in class lectures. Only grouped (aggregate) data will be presented. The data will be securely stored by Dr. Duane Szafron for a minimum of five years.

Benefits and risks. This research can potentially contribute to the advancement of tools for scripting in computer games. There are no foreseeable risks to this study, but if any risks should arise, the researcher will inform the participants immediately. If you should experience any adverse effects, please contact Dr. Duane Szafron and/or Dr. Don Heth immediately.

Contact information. If you have any questions or comments on the study, or if you wish a clarification of rights as a research participant, you can contact Dr. Duane Szafron or the Arts, Science & Law Research Ethics Board at the number and address below.

Dr. Duane Szafron, Ph.D.
Department of Computer Science
University of Alberta
Edmonton, AB T6G 2E9
(780) 492-5468

Dr. Don Heth, Ph.D.
Chair, Arts, Science & Law Research Ethics Board
University of Alberta
Edmonton, AB T6G 2E9
(780) 492-4224

Signatures. Please sign below to indicate that you have read and understood the nature and purpose of the study. Your signature acknowledges the receipt of a copy of the consent form as well as indicates your willingness to participate in this study.

_____ _____

Participant's Signature Date

_____ _____

Researcher's Signature Date

Appendix D - Demographics Form

The original formatting has been altered to fit the dissertation format.

Pre-Experiment Demographics Survey

Participant Number: _____

Age:___
Gender:___
Current Degree and
Major:_____
Year of Study: _____
Years of programming experience: _____

How regularly do you play video games? (check one)
Never ☐ Once a month ☐ Once a week ☐
Several times weekly ☐ Daily ☐

How many computer or console role-playing games (RPGs) have you played? (check one)
0 ☐ 1 ☐ 2 ☐ 3 ☐ 4-6 ☐ ≥7 ☐

Have you played Neverwinter Nights before? (check all that apply)
Never played ☐ Played but never completed ☐ Completed ☐
Created modules ☐

How many scripts have you written using NWScript in the Aurora Toolset? (check one)
0 ☐ 1-9☐ 10-99 ☐ ≥100 ☐

How many patterns have you made in ScriptEase? (check one)
0 ☐ 1-9☐ 10-99 ☐ ≥100 ☐

Appendix E - NWScript Tutorial

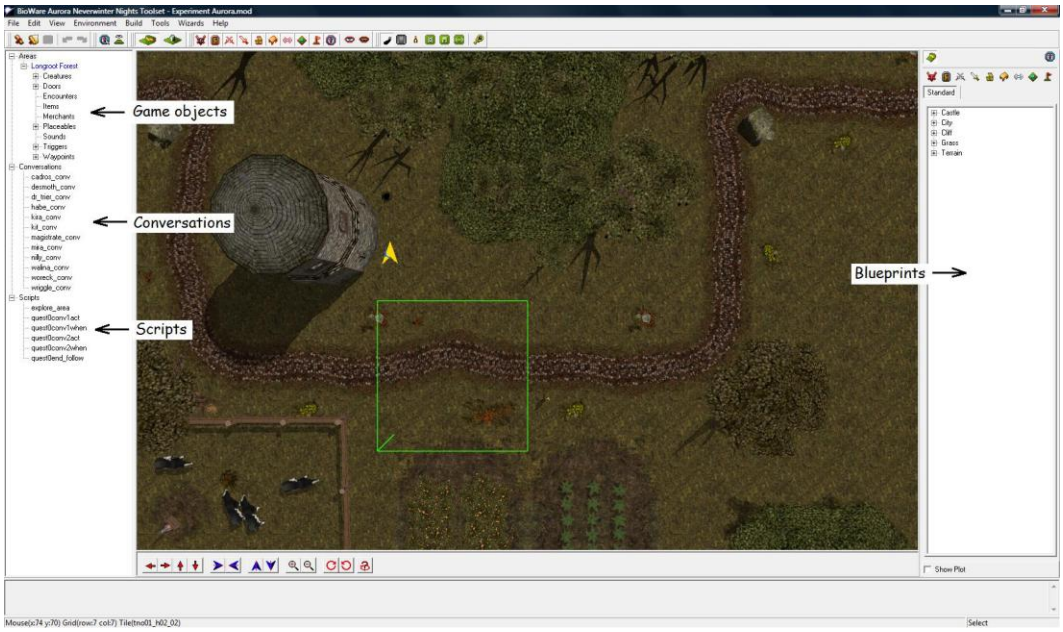
The original formatting has been altered to fit the dissertation format.

How to Script with NWScript and the Aurora Toolset

The Aurora Toolset is a large and powerful tool, but this tutorial will only cover scripting, as that is the focus of the experiment. The relevant GUI controls are highlighted in the screenshot below. Your goal is to write scripts, but you will need to attach them to game objects, blueprints, and conversations.

Loading the Module

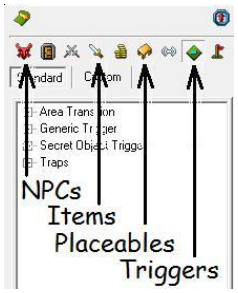
When the experiment begins the proper module and area should already be opened. Should it ever be closed and need to be reopened select the "File" drop-down menu and select "Open". Choose to open the "Experiment Aurora" module. Expand the "Areas" hierarchy on the left sidebar and double click on "Longroot Forest". The area will load and the scripting can proceed.



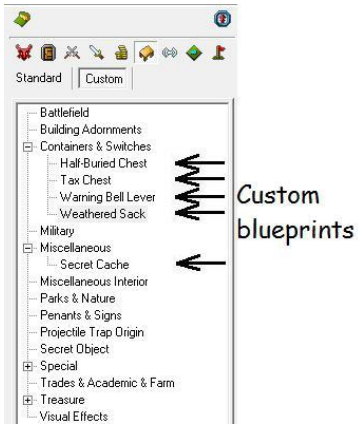
Game Objects, Blueprints, and Resource References

A game object is anything that can be placed on the map that is not terrain: non-player characters (NPCs), items, and placeables (chests) are all game objects. How game objects are referenced in Neverwinter Nights is similar to how objects are referenced in object-oriented programming. A blueprint is like a class. A blueprint is uniquely identified by its resource reference (ResRef) as a class is by its name. Instances of blueprints are called game objects in this tutorial.

Blueprints can fall into several categories, with only creatures, placeables, items, and triggers relevant to the experiment. The following picture to the left shows the buttons to click to access the lists of these blueprints. These categories are further divided into those blueprints that were provided with the game, the standard blueprints, and those that were created for a specific module, the custom blueprints. **This experiment only uses custom blueprints.** Each category of blueprints is arranged in a hierarchy. The blueprints are found at the bottom of the hierarchy.



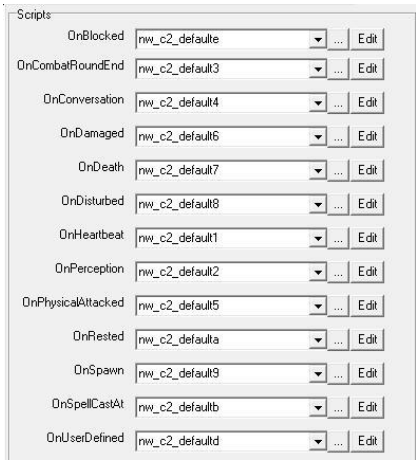
All the custom item blueprints are shown in the picture to the right. They were revealed by clicking the placeable icon, the "Custom" button, the plus sign for "Containers & Switches" folder, and then the plus sign for "Miscellaneous". The name attribute for the blueprint is displayed.



Names and tags are two important attributes for game objects and blueprints. The name is what the player sees when playing the game. A tag is a second name that is invisible to the player. **Scripts refer to game objects by their tags. Tags do not have to be unique. If a script refers to a tag then it will randomly pick a game object with that tag.**

In this module every blueprint has a unique tag. The game objects of that blueprint all use that tag. That is not a problem since **no script in this experiment will require distinguishing between two objects of the same blueprint**.

To view the name and tag of a blueprint right click on the blueprint and select "Edit". The properties window of the blueprint will appear and the "Basic" tab will contain the name and tag.



Events

All scripts occur in response to events. For example, an NPC has a script that describes what happens when that NPC is killed. The picture to the left shows the NPC can have scripts to react to many different events. Each event has a slot that can hold a script. This NPC already has default scripts for each slot. These default scripts can be edited or replaced with a new script.

One of the more unintuitive slots is the

“OnHeartbeat” slot. **A game object's heartbeat is a script that fires every six seconds during the game.**

The “...” button is used to select an existing script. The “Edit” button opens up the NWScript editor so changes can be made to this script.

Scripts can be attached to several different constructs. The relevant constructs are blueprints, game objects, conversations, and module events.

Editing a Blueprint's Scripts

The blueprints can be found in the right sidebar. Click on the icon representing the type of blueprint and click the “Custom” button. Navigate to the bottom of hierarchy to find the module's blueprints. To edit a blueprint's scripts right click on the blueprint and select “Edit”. When the properties window appears, go to the “Scripts” tab. The “...” button is used to select an existing script. The “Edit” button opens up the NWScript editor so changes can be made to this script.

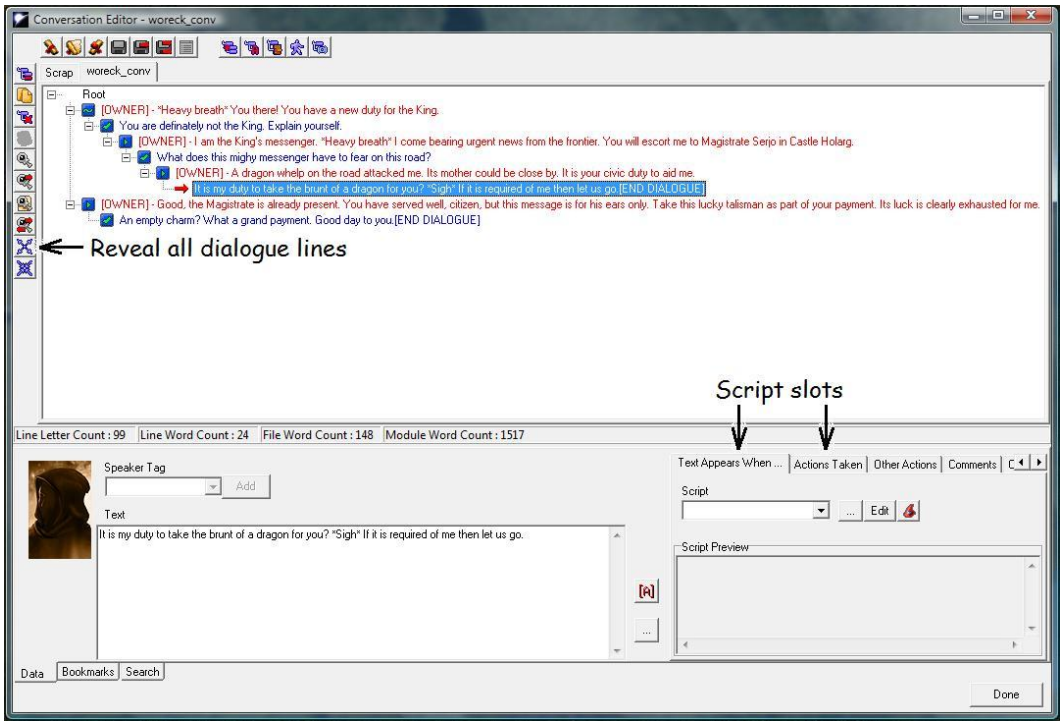
Changing the blueprint's script will not automatically change the scripts of its game object instances. Updating instances will remove any scripts from the game objects and replace them with the blueprint's scripts. To update a blueprint's instances right click on the blueprint and select “Update Instances”. A series of questions concerning the scope of the update will now appear. The questions will be slightly different depending on the type of blueprint being updated. Answer “Yes” to each question with that option. If a screen with check boxes appears then click “Select All” and select “Okay”.

Editing a Game Object's Scripts

All of the game objects in the module are found on the left sidebar of the screen. Right click the game object and select “Properties”. When the properties window appears, select the “Scripts” tab. The “...” button is used to select an existing script. The “Edit” button opens up the NWScript editor so changes can be made to this script.

Conversations

A conversation is a sequence of dialogue lines alternating between an NPC (red) and the PC (blue). One conversation file holds all the dialogue between a single NPC and the PC (player character).



All the conversations present in the module can be found in the left sidebar of the screen. To open a conversation double click the conversation's name. As an example, the conversation file *woreck_conv* contains dialogue between the NPC *Woreck* and the PC. The picture above shows there are two main branches of the conversation: the branch where *Woreck* gives the PC a quest and the branch where *Woreck* thanks the PC for completing the quest.

Scripts are needed to decide which lines of dialogue are shown under which circumstances and what actions are taken when a dialogue line is reached. All conversations start at the “Root” node. The conversation can then proceed to one of the red NPC dialogue lines contained within. The potential dialogue lines will be examined in order from top to bottom. The script in the “Text Appears When ...” slot determines if that dialogue line can be shown. **The first dialogue line that the NPC is allowed to say will be chosen and displayed.**

Red NPC dialogue lines contain a set of blue dialogue lines the PC can choose from. Each PC dialogue line can also have a “Text Appears When ...” script. **Every dialogue line that a PC is allowed to say will be chosen and displayed.**

Whenever an NPC dialogue line is displayed or a PC dialogue line is chosen then the script in the “Actions Taken” slot will be executed.

Editing a Conversation's Scripts

The list of all conversations can be found in the left sidebar. Double click a conversation file to bring up its window. In a conversation window there are

two slots to attach scripts to. These slots can be found under the “Text Appears When ...” tab and the “Actions Taken” tab. The “...” button is used to select an existing script. The “Edit” button opens up the NWScript editor so changes can be made to this script.

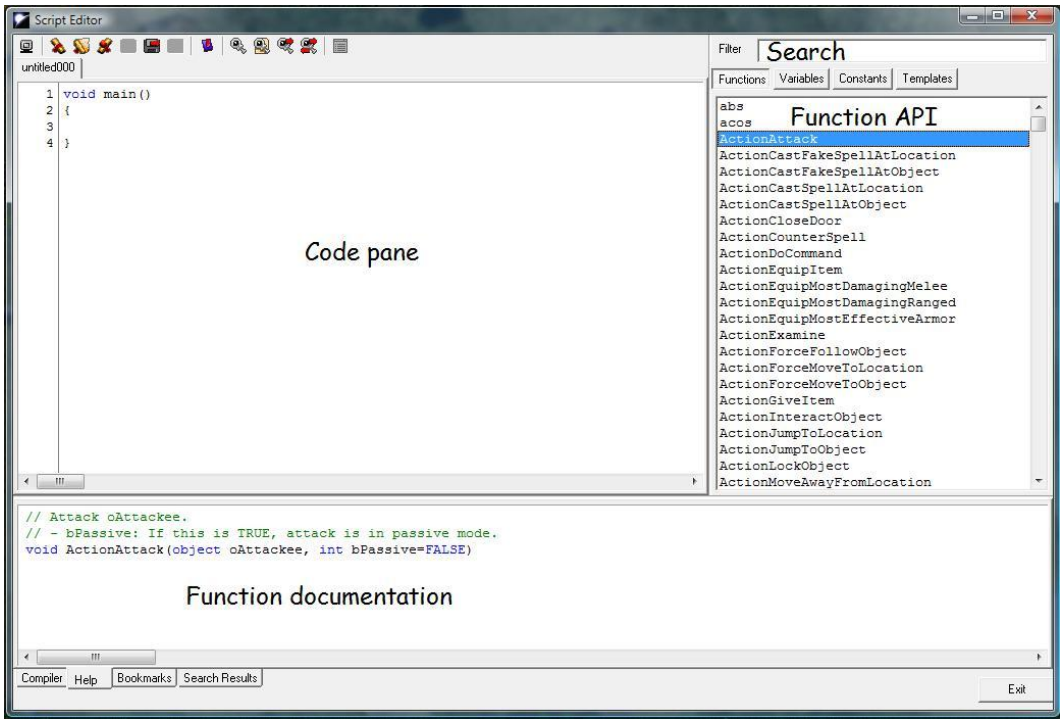
Editing Scripts on the Player Character's Events

All of the events that happen to a PC or to the module itself are found in the module's event list. This list can be found by clicking on the “Edit” drop-down menu and selecting “Module Properties”. When the properties window appears click on the “Events” tab. All of the script slots are now visible. The “...” button is used to select an existing script. The “Edit” button opens up the NWScript editor so changes can be made to this script.

Scripting

Once a script window is open there are several tools at your disposal, most of which are shown in the picture below. The search field can be used to find functions and constants whose name contains a given string. The function API shows all the functions that pass through the search filter. If the function is selected its documentation appears in the bottom pane. If the function is double clicked then that text appears where the cursor is in the coding pane. The code pane is where the scripting code is typed.

The scripting language, NWScript, uses a syntax close to C but with less functionality. **Everything in NWScript is case sensitive**. There are four important data types in NWScript: integers (int), strings (string), objects (object), and actions (action). Integers and strings work exactly like their counterparts in C. **Objects act like another primitive data type. They do not have methods and their data can only be accessed indirectly through functions.** Objects cannot be created or destroyed explicitly; other functions must be called to do that. Actions are commands for game objects. Some actions occur immediately; others will be put in a queue. **Actions can not be used with an '=' operator. Actions can only be created with functions and them immediately used in other functions. The lexicon of all NWScript functions with can be found on the desktop as Lexicon.chm.**



There are a number of features in NWScript that are not found in C. Each script is associated with an object that can be referenced through the OBJECT_SELF constant. For example, if a wolf has a script in its OnKilled slot then that wolf can be referenced through that script's OBJECT_SELF constant. Functions also have optional parameters. The function shown in the picture above has an optional parameter. In that example, the bPassive parameter is optional. “ActionAttack(oAttackee)” and “ActionAttack(oAttackee, bPassive)” are both valid uses of the function. In the case of the former the parameter bPassive is assumed to be false, as indicated in the function's documentation.

Data can be stored and retrieved through the SetLocalInt and GetLocalInt functions, with similar functions for strings. These functions require the object which will hold the data as a parameter. If the object the data is stored on is killed then the data is lost, so it is important to pick the correct object to store the data on. The example quest stores all data on the module object itself since the module can never die.

The script will compile every time it is saved. The results of the compile will be displayed in the bottom bar, where the function documentation usually is. The game can still run even if the script does not successfully compile. In that case the script will not be executed at all.

Finally, if a default script must be edited then new functionality should be added onto the script instead of removing the existing functionality. Do not break what already works in the game in order to making something else work.

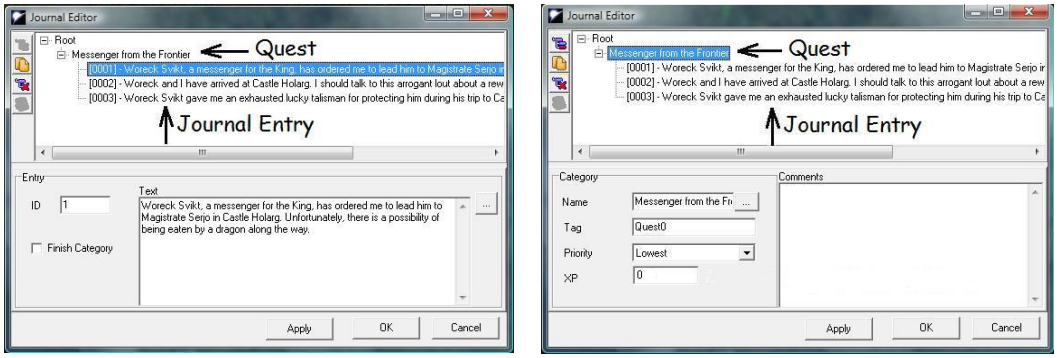
Journals and Quests

Journals are used to remind the PC of what they have done and what they should be doing. Journals are the primary way of measuring progress in a quest. To edit journal entries click on the "Tools" drop-down menu and select "Journal Editor". The window similar to the picture below should appear.

Quests are displayed in a hierarchy three layers deep. The first layer is the root node, which does nothing. The second layer contains the names of the quests. The third layer contains the numbered journal entries for that quest.

A new quest can be created by right clicking the "Root" and selecting "Add". When the quest is selected the window should look like window in the picture below on the left. A quest has a name and a tag. The players see the name of the quest, while scripts refer to the tag of the quest. **Do not use the XP field; it does not work, and will not award the PC any experience.**

A journal entry is created by right clicking a quest and selecting "Add". When a journal entry is selected the window should look like the window in the picture below on the right. The "Text" box in the bottom center of the window is where the text of the journal entry goes. Each journal entry of a quest has an ID number which is used in scripting. **The "Finish Category" check box marks this journal entry as being a final journal entry for the quest.** When that box is checked the journal entry will appear in the "Quests Completed" tab of the journal.



Example Quest

An example quest has already been scripted in the current module. This section will describe the techniques and ideas used in scripting the quest. The text of the journal entries and scripts themselves will not be shown in the tutorial since they can be easily viewed in the module.

The goal of the quest is simple: To lead *Woreck* to Castle Holarg. In a bit more detail, there are three stages to the quest:

1. The PC talks to *Woreck* to receive the quest and *Woreck* begins to follow the PC

2. The PC arrives at Castle Holarg, causing *Woreck* to stop following the PC
3. The PC talks to *Woreck Svikt* to receive a reward

Six scripts are used in this quest:

- *quest0conv1act* is attached to "Actions Taken" of the line "It is my duty to take the brunt of a dragon for you? *Sigh* If it is required of me then let us go." in *woreck_conv*
- *quest0conv1when* is attached to "Text Appears When" of the line "*Heavy breath* You there! You have a new duty for the King." in *woreck_conv*
- *quest0conv2act* is attached to "Actions Taken" of the line "An empty charm? What a grand payment. Good day to you." in *woreck_conv*.
- *quest0conv2when* is attached to "Text Appears When" of the line in "Good, the Magistrate is already present. You have served well, citizen, but this message is for his ears only. Take this lucky talisman as payment. Its luck is clearly exhausted for me." in *woreck_conv*
- *quest0end_follow* is attached to the "OnEnter" event of the *Castle Holarg Entrance* trigger blueprint

There is one more script that is already present in the module but is not related to this quest. The reveal_map script, which reveals the map for the player, is attached to the "OnEnter" event of the area. **The reveal_map script should not be removed since a revealed map will make navigating the game world easier.**

Variables

To track progression through the quest, a set of variables are stored on the module. Each of these variables stores whether a stage has been completed. Other systems can work but this is the one used in the example. While it may seem intuitive to use one variable to store progress using the ID of the journal entry this creates problems with the non-linear nature of quests. Some quests can be completed in a different ordering of stages, in which case tracking each individual stage provides simpler logic.

Journals

A good first step in creating a quest is writing its journal entries using the Journal Editor. Each stage of the quest naturally correspond to a journal entry. The example quest was made with a name of "Messenger from the Frontier" a tag of "Quest0" for easy reference. Three journal entries where then created. The journal entries followed the chronological order of the quest. This is a requirement of NWScript, since a lower numbered journal entry cannot be displayed after a higher number journal entry. In the example the third and final journal entry was marked as a "Finish Category".

Conversations

Conversations play a large role in governing the quest, so they were scripted next. The file *woreck_conv* contains the conversation between *Woreck* and the PC. The conversation has two main branches: the branch where *Woreck* instructs the PC to lead him to Castle Holarg and the branch where *Woreck* thanks the PC for traveling with him. To ensure the branches appear in the correct circumstance a script was assigned to each of their "Text Appears When ..." slots. The script *quest0conv1when* ensures that *Woreck* will only offer the quest if the PC has not already accepted the quest. This is done by simply checking that the first stage of the quest has not been reached yet.

The script *quest0conv2when* ensures that *Woreck* will only thank the PC for completing the quest if the PC has lead *Woreck* to Castle Holarg and has not yet thanked the PC. This done by checking that the second stage of the quest has been completed but the third stage has not.

These scripts at the start of dialogue branches ensure they will appear at the correct times, but they do not advance the quest in any way. A pair of scripts on the end of those branches in the "Actions Taken" slots do that.

The script *quest0conv1act* will fire when the PC has reached the dialogue line where they agree to travel with *Woreck*. That script will give the PC the first journal entry, mark the first stage as completed, and instruct *Woreck* to follow the PC.

The script *quest0conv2act* will fire when the PC has reached the dialogue line where they thank Woreck for the reward given for traveling to Castle Holarg. That script will give the PC the final journal entry, mark the third stage as reached, give gold and experience to the PC, and transfer the reward item from *Woreck* to the PC.

Blueprints

There is still one last problem to solve in the example quest: How does the game know when the PC has reached Castle Holarg? There is a trigger in front of Castle Holarg named the *Castle Holarg Entrance*. The script *quest0end_follow* is attached to "OnEnter" of the *Castle Holarg Entrance* trigger blueprint. The script checks whether *Woreck* is following the PC by checking that the first stage has been completed. If *Woreck* is following the PC then the second stage is marked as completed, the second journal entry is given, and *Woreck* stops following the PC. **Once the blueprint was changed its instance was updated or else the new script would not be applied.**

How to Play Neverwinter Nights

To test the game click on the build drop-down menu and select “Test Module”. Another way to launch the game is to press 'F9'. Be sure to save any changes.

Game Controls

- Move by left clicking open ground
- Rotate the camera by placing the cursor at the edge of the screen or use the arrow keys
- Fight by left clicking an enemy NPC (a creature with a red name)
- Talk by left clicking a friendly NPC (a creature with a blue name)
- Open a container's inventory screen by left clicking it. Double click an item to take it.
- Left click a dialogue option to select it
- Open the PC's inventory screen by pressing 'i'
- Open the journal by pressing 'j'. The “Quests” tab on the left contains active quests while the “Quests Completed” tab in the middle contains completed quests.
- Open the map by pressing 'm'
- Highlight interactive objects by holding 'Tab'

Appendix F - ScriptEase Tutorial

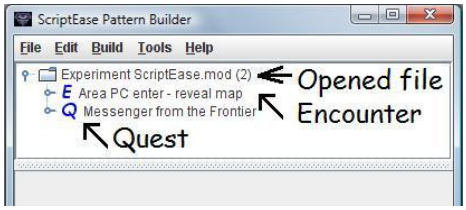
The original formatting has been altered to fit the dissertation format.

How to Script with ScriptEase

The quests of role-playing video games can be broken down into patterns. Common quest patterns include defeating an enemy, retrieving an item, and traveling to a location. ScriptEase provides a catalogue of these patterns and a means of implementing them with scripts. The quest patterns of ScriptEase provide a way of controlling the actions and progression of a quest.

This tutorial will detail how to construct the example 'Escort' quest pattern in the accompanying module. More advanced controls than those needed to construct the example quest pattern will be described afterwards.

Starting ScriptEase



The experiment should begin with ScriptEase already open and the proper module loaded. In case the program exits, ScriptEase can be reopened by double clicking the ScriptEase icon on the desktop. The module is loaded by selecting the “File” drop-down menu and selecting “Open Module ...” and then “Experiment ScriptEase”. **A bug in ScriptEase will occur and you should ask the leading experimenter to fix it.** As the above picture shows, there should already be two things present in the open module: an encounter called “Area PC Enter – reveal map” and an example quest called “Messenger from the Frontier”. Though there are not essential, do not delete either of them. The former allows easier navigation of the game world and the latter provides a useful example.

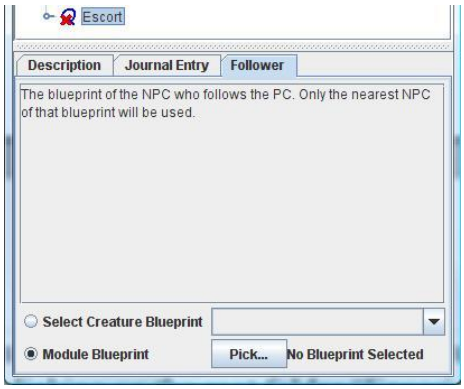
Quest Structure

The idea behind the example quest is simple: The player character (PC) must lead the non-player character (NPC), *Woreck*, to Castle Holarg. The idea behind the quest can be expanded upon to include many details such as what is the reward or how does the game know when the PC has arrived at Castle Holarg. Similarly, programming in ScriptEase starts with the idea behind the quest and gradually expands upon it until all the details have been determined. ScriptEase scripts quests using a hierarchy of constructs. The patterns higher in the hierarchy, such as quest patterns, are more abstract and describe ideas. The patterns lower in the hierarchy, such as encounter patterns, are more concrete and describe actions.

Quest Patterns

A quest pattern represents the idea behind a typical quest found in most role-playing games, usually summed up as a single verb, like

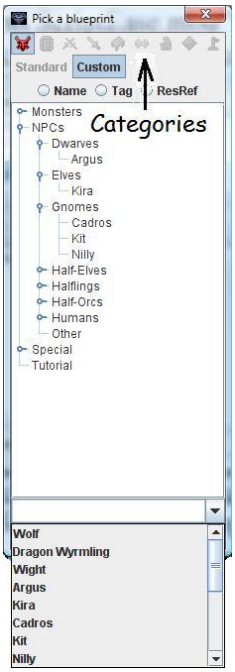
'Assassinate'. The example quest is an 'Escort' quest. The idea of an 'Escort' quest is that the PC must escort an NPC until a goal is reached.



Most operations in ScriptEase are accessible by right clicking. This quest was created by right clicking on the root folder "Experiment ScriptEase.mod" and selecting "**New Specific Quest -> Custom -> Quests -> Escort".** The new quest, represented by a blue 'Q' will appear in the module, similar to the picture on the left. The red 'x' in the bottom-left corner states not enough information has been specified by the user. **A quest with a red 'x' will not work.** Additional information can be specified within the tabs of the pattern or the tabs of other patterns contained within it. The 'Escort' quest pattern has three tabs which provide documentation and options for the users.

The "Description" tab provides the name of the quest pattern, which was set to "Messenger from the Frontier" for the example. A description summarizing the "Escort" quest pattern is also given in that tab.

The "Journal Entry" tab has a field for providing a journal entry if the entire quest fails. The journal entries for a successful quest are entered at the quest point level described later in the tutorial.



While the "Description" and "Journal Entry" tabs are found in all quest patterns, the "Follower" tab is an option specific to this quest. An option in ScriptEase is like a parameter in a programming language; it needs to be set to a value before it can be used. The "Follower" option is set to the blueprint of the NPC who follows the PC. This piece of information is needed for the quest to compile successfully. Every object in the game Neverwinter Nights has a blueprint. All objects of the same type, like wolves, share the same wolf blueprint.

Picking a Blueprint

There are two ways of picking the "Follower" blueprint, as indicated by the radio button at the bottom of the "Follower" tab. The first method is to pick from a drop-down list of appropriate blueprints that have already been used by a pattern higher in the hierarchy. There is nothing higher than a quest pattern though, so that list will be empty for this quest pattern. Quest point patterns and encounter patterns can benefit from this method.

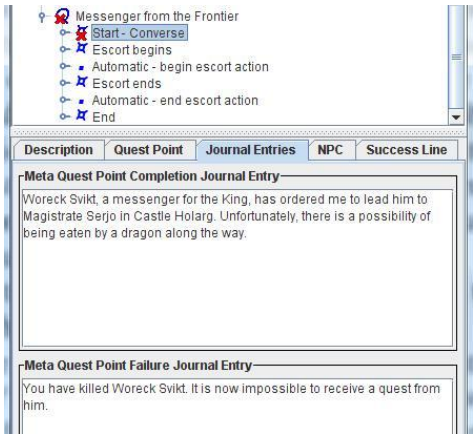
The second method is to use the "Pick..." button to access the blueprint picker, which is shown in the picture to the left. The picker will display a

hierarchy of blueprints organized by category. Navigate a hierarchy to pick a blueprint or choose a blueprint from the drop-down list of all blueprints of the chosen category. Categories that are not appropriate will be grayed out.
Using the second method, “Woreck” was chosen to be the “Follower”. Once someone is chosen to be the “Follower” the red 'x' disappears from the quest. This occurred because several of the patterns contained inside the “Escort” quest pattern referenced the “Follower” and couldn't compile until the “Follower” was set to a NPC. This option mechanism ensures that “Woreck” needs only be selected once to be used in multiple circumstances.

Though the red 'x' has disappeared and there is enough information for the quest to compile, there is not yet enough information to have a perfectly working quest since many ScriptEase patterns have a default that does nothing. For example, the “Start” quest point inside of “Escort” does not nothing by default and that needed to be changed. Those changes are described in the section on quest points below.

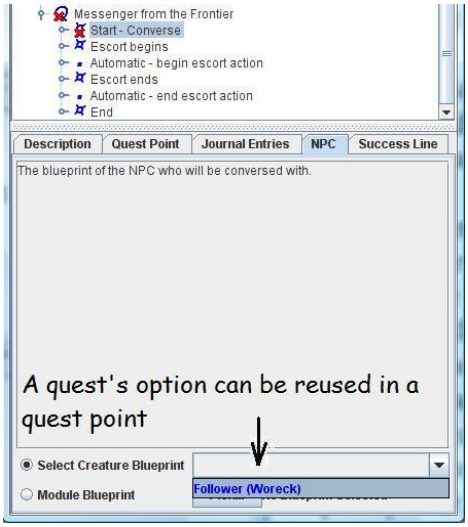
Quest Points and Meta Quest Points

Quest patterns are composed of a sequence of quest point and meta quest point patterns. **Quest points represent key plot events within a quest, such as a conversation or the death of a character. A meta quest point is an abstract version of a quest point representing the different ways that the event could occur.** 'Start' is a meta quest point which represents how a quest can be begin. The meta quest point can be set a quest point like, 'acquiring' with an item, 'conversing' with an NPC, and 'arriving' at an area. The advantage of using meta quest points in quest patterns is flexibility. A smaller catalogue of quest patterns can cover a larger range of quest ideas. Each meta quest point suggests appropriate quest points but they can be set to any quest point.



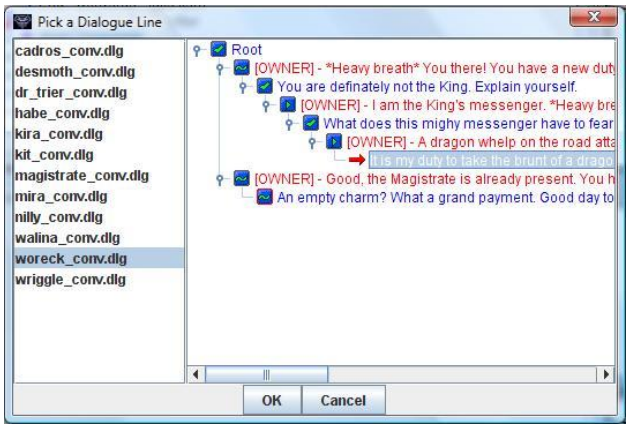
To view the quests and meta quest points of a quest pattern click the blue icon left of the quest. The quest points are marked with a '•' in the picture below. The meta quest points are marked with a '¤'. There are six quest points and meta quest points in the 'Escort' quest pattern The first meta quest point is a “Start” meta quest point. In the example this meta quest point was set to “Converse” since the quest begins through conversation. **Setting a meta quest point to a quest point, in this case a “Converse” quest point, is done by right clicking the meta quest point and selecting “Set Meta Quest Point... -> New Quest Point... -> Custom -> Quests -> Converse”.** Any quest point found at the final list is intended to fill that meta quest point. Unsetting a meta quest point is done by right clicking the

meta quest point and selecting “Set Meta Quest Point... -> Unset”. After the “Start” meta quest point was set to the “Converse” quest point the meta quest point was renamed “Start – Converse”.



If a meta quest point is unset then it does nothing: no journal entries, no generated scripts, absolutely nothing. That quest point is ignored and the quest progresses to the next quest point in the chain. This is different then the 'Automatic' quest point which can still give a journal entry and will generate scripts.

The picture to the left shows the tabs of a quest point and shows the window corresponding to the “Journal Entries” tab. The “Description” tab of a quest point is just like the “Description” tab of a quest: it contains the name of the quest point and a summary of it. The “Quest Point” tab contains information about how this quest point affects the progress of a quest and its contents, which are detailed in a later section of the tutorial. Every quest point has a “Journal Entries” tab. That tab contains journals entries that are shown when that quest point succeeds or fails. The remaining tabs are options for the quest point. A “Converse” requires the “NPC” and “Success Line” options be set, which is why a red 'x' appeared when the “Start” meta quest point was set to “Converse”. The NPC to be conversed with and dialogue line which causes the quest point to succeed must now be specified.



The first option is the NPC who the PC converses with, which is “Woreck”. It is selected the same way as the “Follower” was to the quest. Since the “Follower” was defined higher in the hierarchy than the quest point, the “Follower” option can be reused, as shown in the diagram on the left. Such reuse leads to more easily adaptable patterns, since if the “Follower” was changed to a different NPC that change would propagate to all contained patterns, like “Converse” the quest point in this example, which use that option.

The second option is the “Success Line”, the dialogue line which must be reached for the quest point to succeed. However, the dialogue picker is different

than the blueprint picker. The dialogue picker is shown in the picture below. To understand the picker requires a description of a conversation.

A conversation is a sequence of dialogue lines alternating between an NPC (red) and the PC (blue). One conversation file holds all the dialogue between a single NPC and the PC. For example, the conversation file *woreck_conv* contains dialogue between the NPC *Woreck* and the PC. The picture below shows there are two main branches of the conversation: the branch where *Woreck* gives the PC a quest and the branch where *Woreck* thanks the PC for completing the quest. The line “It is my duty to take the brunt of a dragon for you? *Sigh* If it is required of me then let us go.” of *woreck_conv* corresponds to the PC accepting the quest so it is picked to be the “Success Line”.

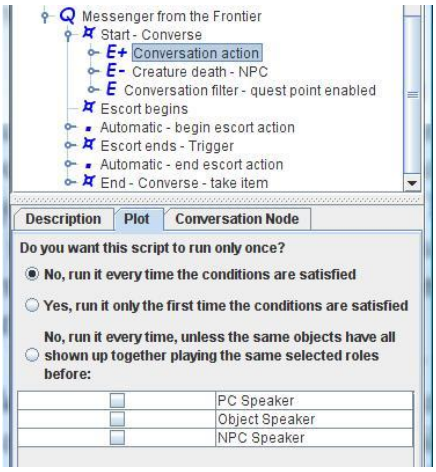
When the options of the “Start - Converse” quest point are set the red 'x' should disappear again. The remaining work with the quest points is simpler. The “Escort Begins” meta quest point should remain unset. This sounds unintuitive but the dialogue line of the “Success Line” of the “Start” meta quest point already states that escort should begin at that point in the story. Thus, the “Automatic – begin escort action” should be the next quest point to occur. The reason the “Escort Begins” exists is that the NPC who gives the quest in “Start” meta quest point may be a different NPC than the one to be escorted. In this example quest those NPCs where the same so the “Escort Begins” doesn't need to do anything.

The “Escort Ends” meta quest point is set to the “Trigger” quest point. That quest point's option of “Trigger” is set to the “Castle Holarg Entrance” trigger. Now when the PC enters the invisible trigger region in the game this quest point will succeed. There is an option for a caption to be spoken by the PC when they enter the trigger, but it remains empty since no caption is desired in this example.

The “End” meta quest point is set to the “Converse – take item” quest point. This quest point scripts that when a specific line a dialogue is spoken *Woreck* should give the PC a *lucky talisman*. The “Giver” option is once again “Woreck”. The “Success Line” is “An empty charm? What a grand payment. Good day to you.” of *woreck_conv*. The “Item” is the “Lucky Talisman”. Finally, the “Quantity” remains at 1. In the “Quest Point” tab the “XP Awarded” field was set to 100 and the “Gold Awarded” field was set to 50. The quest is now almost complete.

Encounters

Just as quests were composed of quest points, a quest point is composed of encounters. **An encounter reacts to an event in the game with a sequence of actions.** For example, an encounter can specify that when a lever is pulled a door opens. A quest point can be expanded to reveal its encounters by clicking the blue icon to the left of the quest point. The result should look similar to the picture to the left.



All encounters are independent of each other; their order does not matter. Encounters are represented by an 'E'. The encounters with a '+' will cause the containing quest point to succeed when the encounter fires. Similarly, an encounter with a '-' will cause a failure. Encounters without a '+' or '-' do not directly affect the success or failure of the quest point. They are placed in the quest point for convenience.

Every encounter has at least two tabs. The “Description” tab has a field for the quest point name and a summary of the encounter. The “Plot” tab has a radio button that specifies how many times this encounter can fire. The default is it fires every time its conditions are met. The remaining tabs are options of the encounter, which work in the same way a quest point's or an quest's options do.

An encounter is composed of condition and action patterns, but you do not need to view or edit any of them for this experiment.

Controlling (Enabling/Disabling) Dialogue

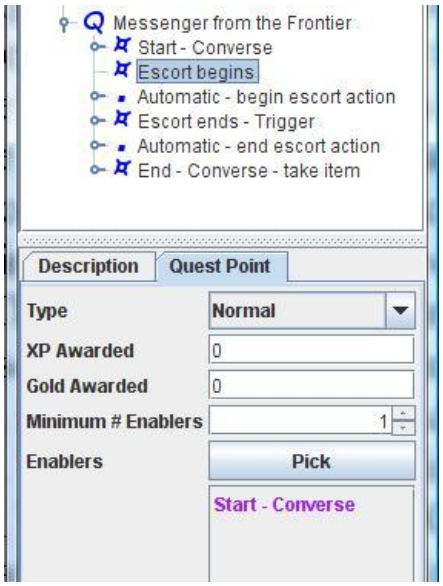
The only thing left to do for the example quest to work is to add an encounter to each of the converse quest points. Currently, the top branch of *woreck_conv* will always appear during a conversation. However, the branch dealing with accepting the quest should only appear before the PC accepts the quest, and similarly, the the branch dealing with rewarding the PC should only appear after the PC reaches Castle Holarg. Thus, the 'accepting' branch should only appear when “Start – Converse” is enabled; a discussion of what it means for a quest point to be enabled is in the 'Quest Point Progress and Operatoins' section. Similarly, the 'reward' branch should only appear while “End – Converse take item” is enabled.

To control the dialogue a “Conversation filter – quest point enabled” encounter must be created for each quest point that involves conversation. To create the encounter for the 'accepting' branch right click the “Start – Converse” quest point. Select “New Specific Encounter -> Custom -> Quests -> Conversations -> Conversation filter – quest point enabled”. The “Dialogue Line” should be set to the line “*Heavy breath* You there! You have a new duty for the King.” in *woreck_conv*. The “Quest Point” option should be set to “Start – Converse”. Create a similar encounter for “End – Converse take item” except the “Dialogue Line” should be “Success Line” is “Good, the Magistrate is already present. You have served well, citizen, but this message is for his ears only. Take this lucky talisman as payment. Its luck is clearly exhausted for me.” in *woreck_conv*, and the “Quest Point” should be “End – Converse take item”. The example quest should now work as desired.

Quest Point Progress and Operations

How Quest Points Track Quest Progression

The progression of a quest is measured in quest points, ordinary and meta. For linear quests sequence of quest point progression is a chain that begins at "Start" and finishes at "End". The example "Escort" quest is linear. The chain is linked together by enablers. The enablers specify the order the quest's events should occur in. For example, the picture below shows the "Escort Begins" quest point is enabled by the "Start" quest point. This makes sense since PC should not be able to escort *Woreck* until they begin the quest.



The first quest point begins enabled. Once a quest point becomes enabled it can either succeed or fail, at which point it stops being enabled. When that quest point succeeds its success journal entry is given and its successor quest point becomes enabled. When the last quest point succeeds the quest is over. If a quest point fails then its failure journal entry is given and nothing new is enabled, possibly severing the chain. **If there is no longer a way for the last quest point to succeed then the entire quest fails** and the quest's failure journal entry is given.

However, the events of a quest do not have to happen in order. Consider a different quest where the PC is hired to kill a monster. The PC would start by conversing with the NPC. Then the PC would kill the monster. Finally, the PC would converse with the NPC again. The quest points of the quest would be

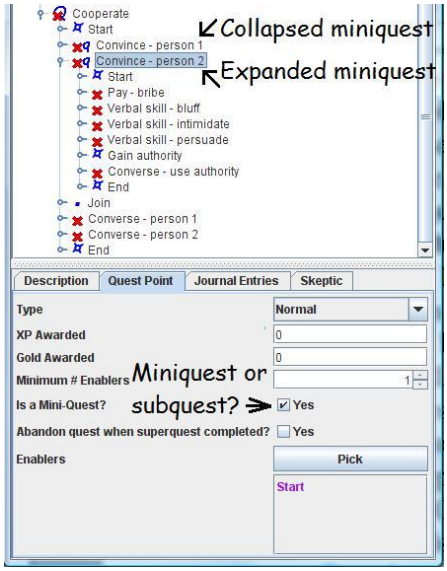
- Start - Converse
- Kill
- End - Converse

However, the quest should still succeed if the PC kills the monster first and then converses with the NPC twice. This is not a problem in ScriptEase since **quest points can be reached without being enabled**. After the first 'Converse' succeeds the 'Kill' would immediately succeed since it was already reached. Since journal entries only appear when the quest point succeeds/fails they are given the order the quest points appear in the quest.

So far linear quests have been discussed. Not all quests are linear though. **A quest point can enable multiple other quest points**, thus creating multiple branches for the quest to progress in. Similarly, **a quest point can be enabled by multiple other quest points**. Consider a quest where the PC must either converse

with an NPC or kill that NPC. The “Start” would **enable** both “Converse” and “Kill”, while the “End” would be **enabled by** both “Converse” and “Kill”. The number of enables required by the “End” would determine how many of the preceding quest point need to succeed. One enable requires either “Converse” or “Kill” to succeed, while two enables would require both to succeed. The list of all enablers for a quest point is found in its “Quest Point” tab.

To change the number of enablers of a quest point go to its “Quest Point” tab shown in the picture to the left. The “Minimum # Enablers” field is located on this pane. To change **which** quest points enable the current one click on the “Pick” button. A window will pop up where you can select which quest points will be enablers. A quest point cannot enable itself.



Some quests are complex enough to have **subquest**s, which are quests within quests. An example would be the 'Cooperate' quest shown in the picture below. In a 'Cooperate' quest two NPCs must be convinced to cooperate. 'Convince' is itself a quest. In the 'Cooperate' quest a 'Convince' subquest is a single quest point. When the 'Convince' subquest succeeds/fails the 'Convince' quest point of 'Cooperate' succeeds/fails.

By using a check box on the “Quest Point” tab a subquest can be turned into a miniquest. **All of a miniquest's journal entires appear under the same heading as the superquest's.** A subquest's journal entries appear under a different heading than the superquest's. In ScriptEase subquest is represented by a 'Q' and a miniquest by a 'q'.

Changing a Quest's Structure

Setting a meta quest point is not the only thing that can change the structure of a quest. Quest points can be reordered, replaced, inserted, and deleted with in a quest.

Quest points can be reordered by dragging a quest point into its containing quest. This action will place the dragged quest point at the bottom of the list of quest points. **Reordering quest points does change how the quest progresses. The first quest point in the list will always function as the starting quest point and will be begin enabled, and the last quest point will always function as the ending quest point. When the ending quest point succeeds the quest succeeds. Any quest point (meta or ordinary) can be replaced with another quest point**. **A replacing quest point will keep the enablers, gold and XP awards, and journal entries from original quest point.** Replacing a quest point is done by

right clicking it and selecting “Replace With...”. The menus then branch to allow you to select any ordinary quest point, meta quest point, and subquest to replace the current quest point with.

Removing a quest point is easy: right click the quest point and select “Delete”. **When a quest point is deleted its enabler links are broken.** All quest points that were enabled by the deleted quest point must now be enabled by another quest point.

To make a new quest point right click on the quest and select “New Specific Quest Point -> Custom -> Quests ->” and then the desired quest point. To make a new meta quest point or new subquest select and “New Meta Quest Point” or “New Subquest” instead of “New Specific Quest Point”. The new quest point will appear at the bottom of the quest. The quest points should now be reordered in the quest so they appear in the correct positions. The enablers of the new quest point must be set as well.

Quest points can also be inserted after another quest point. This does not break the enabler chain. If the chain was originally A -> C and 'B' is inserted after 'A' the chain automatically becomes A -> B -> C. To insert a quest point right click the original quest point and select “Insert...” and navigate to find the quest point, meta quest point, or subquest to be inserted.

Playing Neverwinter Nights

ScriptEase will automatically generate the scripting code for Neverwinter Nights if the “File” drop-down menu is picked and “Save Module and Compile” is selected. A window will appear describing the compiling process. If it is successful then the next step is to run this module in Neverwinter Nights. Click the “Build” drop-down menu and select “Load and Play Module in NWN” to run the module or press 'F9'.

Game Controls

- Move by left clicking open ground
- Rotate the camera by placing the cursor at the edge of the screen or use the arrow keys
- Fight by left clicking an enemy NPC (a creature with a red name)
- Talk by left clicking a friendly NPC (a creature with a blue name)
- Open a container's inventory screen by left clicking it. Double click an item to take it.
- Left click a dialogue option to select it
- Open the PC's inventory screen by pressing 'i'
- Open the journal by pressing 'j'. The “Quests” tab on the left contains active quests while the “Quests Completed” tab in the middle contains completed quests.
- Open the map by pressing 'm'
- Highlight interactive objects by holding 'Tab'

Appendix G - Quest Instructions

The original formatting has been altered to fit the dissertation format.

Quest 1: Lighthouse Inspection

Summary: The PC is hired by Wriggle to inspect the lighthouse and report back.

Relevant locations:

- The NPC *Wriggle* is by the shady tree map marker and speaks the *wriggle_conv* conversation
- The *Lighthouse Region* trigger surrounds the lighthouse at the lighthouse map marker

Tasks:

- After the PC accepts the quest by reaching the dialogue line "That is a full bag. I will gladly do this task." of *wriggle_conv* the dialogue line "Uggghhh it is too hot today." of *wriggle_conv*, which offers the quest, no longer appears.
- Once the dialogue line "That is a full bag. I will gladly do this task." of *wriggle_conv* is spoken the PC receives a quest "Lighthouse Inspection" and its first journal entry "The lazy engineer, Wriggle Broadfoot, wants me to survey the lighthouse for damage."
- When all of the following conditions are met:
  - The PC enters the *Lighthouse Region* trigger
  - The dialogue line "That is a full bag. I will gladly do this task." of *wriggle_conv* conversation is spoken

  then the following actions occur:
  - The PC receives the journal entry "I am no expert, but the lighthouse appears to be in good condition. I should return to Castle Holarg and tell Wriggle Broadfoot the good news."
  - The PC speaks "Hmmm ... everything looks fine here, no cracks" in text above their head
- The dialogue line "*Snore* Huh? What? Oh, there are you are." of *wriggle_conv* should only appear after the PC has entered the *Lighthouse Region* trigger, and it should disappear again when the dialogue line "*Growl* You have won this time." of *wriggle_conv* is spoken.
- Once the dialogue line "*Growl* You have won this time." of *wriggle_conv* is spoken the following actions occur:
  - The quest is over and the PC receives its final journal entry "I did Wriggle Broadfoot's work for him and he cheated me on the payment. I will get my revenge another time."
- The PC receives 25 gold and 100 experience (XP)

Quest 2: The Lost Diamond

Summary: The PC finds a treasure map which leads to the Clautman Diamond.

Relevant locations:

- The *treasure map* item is inside the *half-buried chest* placeable at the shipwreck map marker
- The *Clautman Diamond* item is inside the *weathered sack* placeable at the tower ruins map marker

Tasks:

- When the PC acquires the *treasure map* inside the *half-buried chest* the PC receives a quest "The Lost Diamond" and its first journal entry "What a lucky day; I found a treasure map. It indicates the Clautman Diamond is buried in the ruins of a watchtower."
- When the following conditions are met:
    - The PC acquires the *Clautman Diamond* from the *weathered sack*
    - The PC has acquired the *Treasure Map*

then the following actions occur

    - The quest is over and the PC receives its final journal entry "I have found the massive Clautman Diamond. It must be worth a fortune, maybe even 2 fortunes."
    - The PC receives 150 experience (XP)
- If the PC acquires the *Clautman Diamond* first then no journal entries appear until the *treasure map* is acquired

Quest 3: Dragon Disposal

Summary: Habe pleads for the PC to get rid of a dragon wyrmling. The dragon can be killed or driven off by using a lever to ring the warning bell.

Relevant locations:

- The NPC *Habe*, who is at the north edge of Jurenglade, holds the *pie* item and speaks the conversation *habe_conv*
- The *warning bell lever* placeable is by the watchtower at the watchtower map marker
- The NPC *dragon wyrmling* is at the dragon's lair map marker

Tasks:

- After the dialogue line "So, I need to pull a lever or kill a baby dragon. This sounds too easy. I will be back." of the conversation *habe_conv* is spoken the dialogue line "Greetings. Care to do an old farmer a favour?" of *habe_conv* no longer appears.
- Once the dialogue line "So, I need to pull a lever or kill a baby dragon. This sounds too easy. I will be back." of *habe_conv* is spoken the PC receives a quest "Dragon Disposal" and its first journal entry "There is a dragon wyrmling outside of Jurenglade threatening the chickens and Habe Ugren wants me to get rid of it. I can either kill it or flip the warning bell lever to drive it off."
- The PC receives the journal entry "I have killed the baby dragon. I almost regret it but it was a menace. I should inform Habe Ugren." when all of the following conditions are met:
    - The *dragon wyrmling* has died
    - The dialogue line "So, I need to pull a lever or kill a baby dragon. This sounds too easy. I will be back." of *habe_conv* has been spoken
- The PC receives the journal entry "The warning bell is ringing loudly. Hopefully, it will drive the dragon away or summon the soldiers to defend the village. Habe Ugren will want to hear this good news." when all of the following conditions are met:
    - The PC flips the *warning bell lever*
    - The dialogue line "So, I need to pull a lever or kill a baby dragon. This sounds too easy. I will be back." of *habe_conv* has been spoken
- The dialogue line "Have you rid the countryside of the dragon?" of *habe_conv* should only appear after all the following conditions are met:
    - The *Dragon Wyrmling* is dead or the *warning bell lever* has been flipped.

    - The dialogue line “So, I need to pull a lever or kill a baby dragon. This sounds too easy. I will be back.” has been spoken

  and it should disappear again when the dialogue line “Mmm my favourite kind of pie.” of *habe_conv* has been spoken.
- Once the dialogue line “Mmm my favourite kind of pie.” of *habe_conv* has been spoken the following actions occur:
    - *Habe* gives the PC the *pie* from his inventory
    - The quest is over and the PC receives its final journal entry “Jurenglade's chickens are now safe from the baby dragon, and Habe Ugren has given me a tasty pie. It was a good day.”
    - The PC receives 500 experience (XP)
- If the PC kills the *Dragon Wyrmling* or flips the *warning bell lever* before talking to *Habe* then the PC must still talk to Habe twice: once to get the quest and another to finish the quest

Quest 4: Save Jurenglade from the Undead

Summary: Cadros hires the PC to defeat the zombies attacking Jurenglade.

Relevant locations:

- The NPC *Cadros*, who is at the north edge of Jurenglade, holds the *bear pelt* item and speaks the *cadros_conv* conversation
- The *zombie NPCs* are at the arena map marker

Tasks:

- After the dialogue line "Enough history! I will go and bring them their final death." of *cadros_conv* is spoken the dialogue line "You look like the capable sort. Could you help our village in its hour of need?" of *cadros_conv* no longer appears.
- Once the dialogue line "Enough history! I will go and bring them their final death." of *cadros_conv* is spoken the PC receives a quest "Save Jurenglade from the Undead" and its first journal entry should appear: "Cadros Basker told me that 6 zombies are attacking the town. I should go to the unholy ground of the arena in the northern woods and kill them."
- The PC receives a journal entry "I have killed all the zombies that haunt the arena. I should report the good news to Cadros Basker in Jurenglade." once the following conditions are met:
    - The dialogue line "Enough history! I will go and bring them their final death." of *cadros_conv* has been spoken
    - 6 *zombies* are dead
- The dialogue line "Such a foul stench! You have fought the zombies." of *cadros_conv* should only appear after both of the following conditions have been met:
    - The dialogue line "Enough history! I will go and bring them their final death." of *cadros_conv* has been spoken
    - 6 *zombies* have been killed

  and it should disappear again after the dialogue line "It will do. Goodbye." of *cadros_conv* has been spoken
- Once the dialogue line "It will do. Goodbye." of *cadros_conv* has been spoken then the following actions occur:
    - *Cadros Basker* gives the PC the *bear pelt* from his inventory
    - The quest is over and the PC receives its final journal entry "Jurenglade is safe from the undead, and I have been paid with an expensive bear pelt."
    - The PC receives 300 experience (XP)
- If the PC kills the *zombies* before talking to *Cadros* then the PC must still talk to *Cadros* twice: once to get the quest and another to finish the quest

Quest 5: Medicine Run

Summary: Doctor Trier hires the PC to collect trelion leaves from Walina.

Relevant locations:

- The NPC *Doctor Trier* is by the Castle Holarg map marker and speaks the *dr_trier_conv* conversation
- The NPC *Walina*, who is at the north edge of Jurenglade, holds the *trelion leaves* item and speaks the *walina_conv* conversation

Tasks:

- After the dialogue line "Okay, okay, I am going." of *dr_trier_conv* is spoken the dialogue line "Traveler, I need your help." of *dr_trier_conv* no longer appears.
- Once the dialogue line "Okay, okay, I am going." of *dr_trier_conv* is spoken the PC receives a quest "Medicine Run" and its first journal entry "An outbreak of ecoplamenia requires trelion leaves to cure. The rather somber Doctor Trier wants me to collect these leaves from Walina Selsea in Jurenglade."
- The dialogue line "What do you want? Please tell me you can fix a wagon." of *walina_conv* will only appear when the dialogue line "Okay, okay, I am going." of *dr_trier_conv* has been spoken, and should disappear when the dialogue line "I have to go now." of *walina_conv* has been spoken.
- When the dialogue line "Not that disease again. Here, take them. Blasted wagon *kick*. I know Doctor Trier will pay me back." of *walina_conv* is spoken *Walina* gives the PC the *trelion leaves* from her inventory.
- When the PC acquires the trelion leaves they receive the journal entry "I have obtained the trelion leaves from Walina Selsea. I must quickly deliver them to Doctor Trier at Castle Holarg."
- The dialogue line "I can smell the trelion leaves on you. Please give them to me." of *dr_trier_conv* should only appear when the PC has the *trelion leaves*, and it should disappear when the dialogue line "*Snatch* Goodbye. I have lives to save." of *dr_trier_conv* has been spoken.
- Once the dialogue line "*Snatch* Goodbye. I have lives to save." of *dr_trier_conv* has been spoken following actions occur:
    - The PC gives *Doctor Trier* the *trelion leaves* from their inventory
    - The quest is over and the PC receives its final journal entry "Doctor Trier has the trelion leaves she needs to cure the sick in Castle Holarg, but she refused to give me a reward."
    - The PC receives 200 experience (XP)

Quest 6: A Taxing Situation

Summary: Nilly wants the PC to frame the tax collector Argus by planting liquid rainbow in his tax chest.

Relevant locations:

- The NPC *Nilly* is by the wharf map marker and speaks the *nilly_conv* conversation
- The *liquid rainbow* item is inside the *secret cache* placeable by the watchtower map marker
- The *tax chest* placeable is near the shady tree map marker
- The NPC *Magistrate Serjo* is near the Castle Holarg map marker

Tasks:

- After the dialogue line "From past experience ..." in the conversation *nilly_conv* is spoken the dialogue line "Hi there cutey. Are you interested in some dirty work?" of *nilly_conv* no longer appears.
- Once the dialogue line "From past experience ..." of *nilly_conv* is spoken the PC receives a quest "A Taxing Situation" and its first journal entry "Nilly Basker has asked me to frame the tax collector Argus Ritker for drug smuggling. She suggests I obtain some liquid rainbow. I should learn where I can find it."
- Once the dialogue line "This sounds like fun. I will do it." of *nilly_conv* is spoken the PC receives a journal entry "Nilly Basker says I can find some liquid rainbow in a secret cache by the watchtower."
- The PC receives the journal entry "I have acquired some liquid rainbow. Now I need to put it in Argus Ritker's tax chest. He should be close to Castle Holarg." when all of the following conditions are met:
    - The PC acquires *liquid rainbow*
    - The dialogue line "From past experience ..." of *nilly_conv* has been spoken.
- The PC receives the journal entry "The setup is complete. All I have to do is tell the Magistrate of Castle Holarg about the liquid rainbow in the tax chest and then I can watch the fireworks." when all of the following conditions are met:
    - The *tax chest* receives *liquid rainbow*
    - The dialogue line "From past experience ..." of *nilly_conv* has been spoken.
- The dialogue line "What do you want, peasant?" of *magistrate_conv* should only appear when all of the following conditions are met:
    - The *tax chest* contains *liquid rainbow*
    - The dialogue line "From past experience ..." of *nilly_conv* has been spoken.

and it should disappear when the dialogue line “Then do not let me delay you from checking his tax chest.” of *magistrate_conv* has been spoken.

- When the dialogue line of “Then do not let me delay you from checking his tax chest.” of *magistrate_conv* has been spoken the PC receives a journal entry “The Magistrate was fuming at Argus Ritker's 'guilt'. The poor sod. I should tell Nilly Basker of my success. She was at the wharf the last time I talked to her.”
- The dialogue line “I hope you have good news. I am downright giddy with anticipation.” of *nilly_conv* should only appear when the dialogue line “Then do not let me delay you from checking his tax chest.” of *magistrate_conv* has been spoken, and it should disappear when the dialogue line “It was my pleasure.” of *nilly_conv* has been spoken.
- Once the dialogue line “It was my pleasure.” of *nilly_conv* has been spoken the following actions occur:
    - The quest is over and the PC receives its final journal entry “Nilly Basker was overjoyed that her scheme succeeded. She was positively giddy.”
    - The PC receives 400 experience (XP)
- If the PC acquires the *liquid rainbow* or places it in the *tax chest* before talking with *Nilly* the PC must still talk to *Nilly* before talking to *Magistrate Serjo*.

Quest 7: The Bandits' Hostage

Summary: The PC must save Desmoth from bandits and return him to his mother, Kira.

Relevant locations:

- The NPC *Kira*, who is by the potato field in Jurenglade, holds the *ransom note* item and speaks the *kira_conv* conversation
- The NPC *Mira* is by the orchard in Jurenglade and speaks the *mira_conv* conversation
- The *bandit camp entrance* trigger is between Jurenglade map marker and the bandit camp map marker
- The NPC *Desmoth* is at the bandit camp map marker and speaks the *desmoth_conv* conversation

Tasks:

- After the dialogue line “I will bring him back too.” of *kira_conv* is spoken the dialogue line “Help! Help! My son has been kidnapped. My Desmoth is gone.” of *kira_conv* no longer appears.
- Once the dialogue line “I will bring him back too.” of *kira_conv* is spoken the follow happens:
    - PC receives a quest “The Bandits' Hostage” and its first journal entry “A hysterical Kira Ugren asked me to save her son Desmoth from bandits. I should talk with Mira Selsea; she knows where the bandits came from.”
    - The PC reveive the *ransom note* from *Kira Ugren's* inventory.
- The dialogue line “What can I do for you, stranger?” of *mira_conv* only appears when the dialogue line “I will bring him back too.” of *kira_conv* has been spoken, and it should disappear when the dialogue line “Then I will distribute righteous vengence for him too.” of *mira_conv* has been spoken.
- When the dialogue line “Then I will distribute righteous vengence for him too.” of *mira_conv* is spoken the PC receives a journal entry “Mira Selsea spotted the bandits fleeing south-west into the woods. I should look for Desmoth there.”
- The PC receives the journal entry “I have found poor Desmoth. He is in a camp full of bandits. I must talk him into following me.” when all of the following conditions are met:
    - The PC enters the *bandit camp entrance* trigger
    - The dialogue line “I will bring him back too.” of *kira_conv* has been spoken.
- The dialogue line of “AHHH! Please do not hurt me!” of *desmoth_conv* appears when the PC has entered the *bandit camp entrance* trigger, and

disappears when the dialogue line of “Follow me.” of *desmoth_conv* has been spoken.

- When all of the following conditions are met
  - The PC has entered the *bandit camp entrance* trigger
  - The dialogue line “Follow me.” of *desmoth_conv* has been spoken

  then the following actions occur:
  - *Desmoth* follows the PC
  - The PC receives the journal entry “The traumatized Desmoth is following me back to his mother in Jurenglade's farm. I should not waste any time.”
- When the following conditions are met for the first time
  - The PC is within 8.0 metres of the *Kira*
  - The dialogue line “Follow me.” of desmoth_conv has been spoken

  then the following actions occur:
  - *Desmonth* stops following the PC
  - The PC receives the journal entry “I have returned young Desmoth to his mother, Kira Ugren. I should talk to her about the reward.”
- When the dialogue line of “Desmoth! You brought my Desmoth back. How can I ever thank you?” of *kira_conv* only appears when the following conditions are met:
  - The PC is within 8.0 metres of *Kira*
  - The dialogue line “Follow me.” of *desmoth_conv* has been spoken

  and it should disappear when the dialogue line “I looted the bandits' camp. ...” of *kira_conv* has been spoken.
- Once the dialogue line “I looted the bandits' camp. ...” of *kira_conv* has been spoken these actions occur:
  - the quest is over and the PC receives its final journal entry “ Desmoth is back with his mother and I have the bandits' loot. Everybody is satisfied, well except for the bandits.”
  - The PC receives 700 experience (XP)

Quest 8: Wolfbane

Summary: Kit hires the PC to kill the pack of 5 wolves that threaten her cows.

Relevant locations:

- The NPC *Kit* is at the cow pen in Jurenglade and speaks the *kit_conv* conversation
- The *wolf* NPCs are at the wolves' den map marker

Tasks:

- After the dialogue line "I will return when I wear their hides as boots." of *kit_conv* is spoken the dialogue line "Hello there. Come to admire our farm? It is not much, but it is ours." of *kit_conv* no longer appears.
- Once the dialogue line "I will return when I wear their hides as boots." of *kit_conv* is spoken the PC receives a quest "Wolfbane" and its first journal entry "Kit Basker is having trouble with a pack of 5 wolves. Enough trouble that she wants them dead. I can find the wolves in the eastern woods."
- The PC receives a journal entry "I have killed the wolves and turned their pelts into a new pair of boots. My feet will be warm this winter. I should show Kit Basker back in Jurenglade." once the following conditions are met:
    - 5 *wolves* are dead
    - The dialogue line "I will return when I wear their hides as boots." of *kit_conv* has been spoken
- The dialogue line "Since my poor cows have not met your blade, I can only hope those wolves have." of *kit_conv* should only appear after all of the following conditions have been met:
    - 5 wolves have been killed
    - The dialogue line "I will return when I wear their hides as boots." of *kit_conv* has been spoken

  and it should disappear when the dialogue line "Thank you." of *kit_conv* has been spoken.
- Once the dialogue line "Thank you." of *kit_conv* has been spoken then the following actions occur:
    - The quest is over and the PC receives its final journal entry "Kit Basker was pleased to learn I have killed the wolf pack. Another honest day's work is done."
    - The PC receives 200 gold (XP)
- If the PC kills the *wolves* before talking to *Kit* then the PC must still talk to *Kit* twice: once to get the quest and another to finish the quest

Appendix H - Quest Comparison

The original formatting has been altered to fit the dissertation format.

Quest Comparison

Participant number: _____
Quest number: _____

Enter a time for the first two statements and a number for all further statements.

	NWScript	ScriptEase
Starting time for quest work		
Ending time for quest work		
Number of compiles after the first in-game test run		
Number of in-game test runs		
Number of bugs encountered with the tool		

Respond to each statement by marking one of the five choices

	Strongly Agree	Slightly Agree	Neutral	Slightly Disagree	Strongly Disagree
ScriptEase was easier to use than NWScript					
ScriptEase allowed me to make quests faster than NWScript					
ScriptEase allowed me to find and fix errors faster than NWScript					

Appendix I - Overall Tool Comparison

The original formatting has been altered to fit the dissertation format.

Overall Tool Preference

Participant number: _____

Respond to each statement by marking one of the five choices and provide a brief explanation for your choice.

	Strongly Agree	Slightly Agree	Neutral	Slightly Disagree	Strongly Disagree
Overall, I preferred using ScriptEase instead of NWScript to create the quests found in this study.					

Why?

_____

_____

_____

	Strongly Agree	Slightly Agree	Neutral	Slightly Disagree	Strongly Disagree
Overall, I would prefer to use ScriptEase instead of NWScript to create quests in a module of my own design.					

Why?

_____

_____

_____