University of Alberta

Quest Patterns in Computer Role-Playing Games

by

Curtis Aaron Onuczko ©

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

Edmonton, Alberta
Fall 2007

Your file Votre référence
ISBN: 978-0-494-33320-4
Our file Notre référence
ISBN: 978-0-494-33320-4

Sometimes my plot lines are so convoluted, I get calls from friends at 3 am saying; you SOB, you'll never pull this one off.

– Clive Cussler.

Abstract

Specifying the plot in Computer Role-Playing Games (CRPGs) requires a large number of scripts that are difficult to program, track and maintain. This work introduces quest patterns, a high level and intuitive way to structure the plot in CRPGs. Quest patterns are recurring themes (patterns) that can be adapted to suit the game author's needs. Quest patterns have been added to ScriptEase, a generative design pattern tool that can automatically turn pattern specifications into scripts. CRPGs often include simple plots, called side-quests, that are independent from the main plot. Side-quests are important, as they add value to the open-world appeal of the game. This work introduces a tool to aid in the rapid creation of side-quests. Using objects from the CRPG, the tool creates outlines of side-quests that can be used in the game.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Narrative story-telling is an art that has been around for thousands of years. While stories were originally delivered orally, they have also been delivered through writing in literature and visually and aurally in plays and movies. It is a craft which has become quite refined through the generations. In narrative story-telling, the plot is a sequence of linear events. The listener or reader has no influence as to how the story unfolds.

Interactive stories are a newer form of media that have been popularized through computer games. Earlier forms of interactive stories exist, such as the Choose Your Own Adventure line of novels [19] and pen and paper role playing games (e.g. DUNGEONS AND DRAGONS [9]). In interactive stories the reader, or player, becomes the central character and can take actions that influence the story. This allows the player to have a direct influence upon the outcome of the story. Since an interactive story author must take into consideration all possible choices that the player may take at any time during the story, it makes the plot of an interactive story much more complex than a plot in a narrative story.

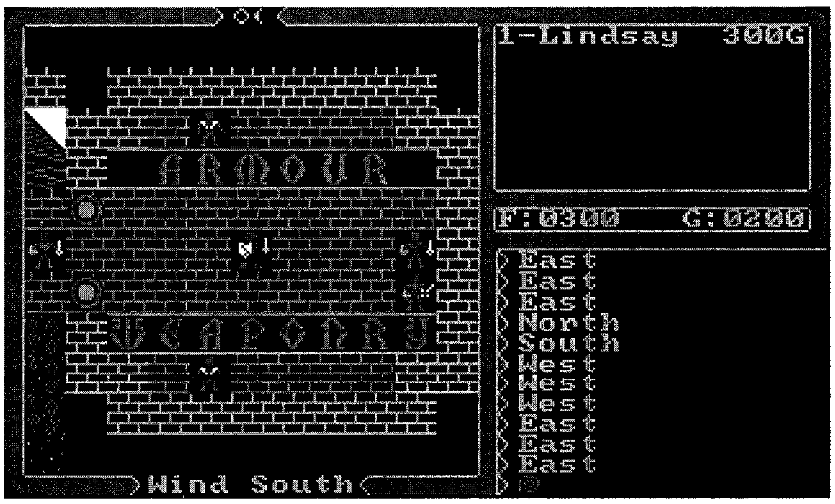


Figure 1.1: Screenshot of ULTIMA IV: QUEST OF THE AVATAR, by ORIGIN Systems, Inc.

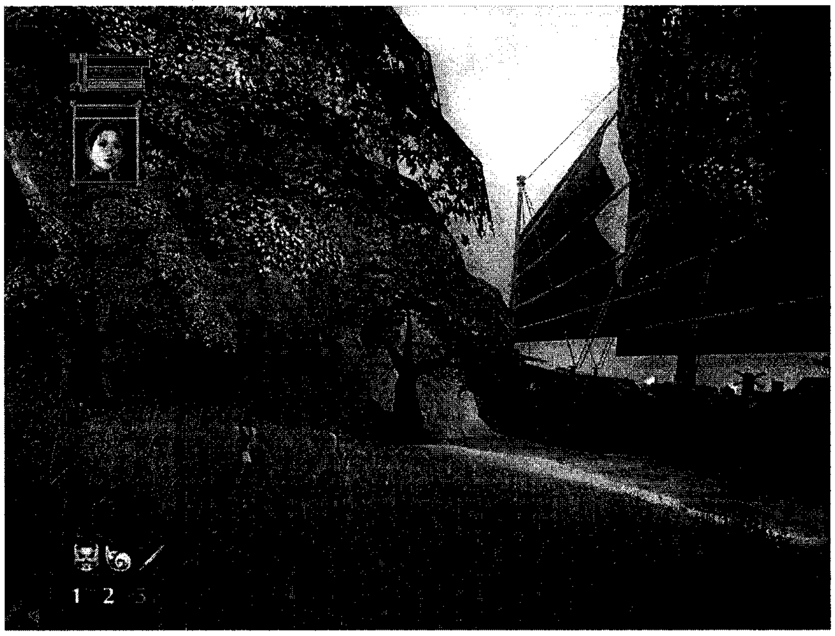


Figure 1.2: Screenshot of JADE EMPIRE, by Bioware Corp.

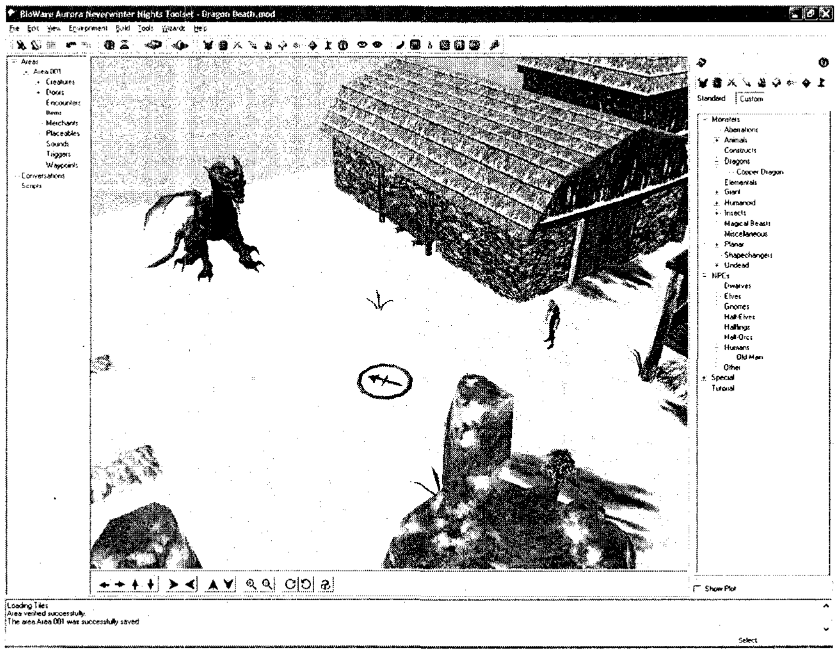
Computer games have matured quickly in recent years. The Computer Role-Playing Game (CRPG) ULTIMA IV: QUEST OF THE AVATAR by ORIGIN Systems, Inc., (1985) has less than 20 people credited to its development team [26]. Figure 1.1 shows a screen shot of ULTIMA IV; the characters are simple icons and the game is played in a two-dimensional world. The dialogue is text-based and consists of simple statements made by the non-player characters (NPCs) while the player character (PC) only gets to respond *yes* or *no* to questions. The plot consists of a single story line. There are no optional adventures for the PC to participate in; the PC can only take actions to advance the main story line. Contrast this with the CRPG JADE EMPIRE [10] by Bioware Corp. [1], (2005), which has hundreds of people credited to its development team [11]. Figure 1.2 shows a screen shot of JADE EMPIRE; the characters look realistic and game-play takes place in a three-dimensional world. Professional voice actors voice all of the NPC dialogue. Conversations consist of many levels of exchanges between the PC and the NPC that can change depending on the PC's current choices in the conversation and previous interactions with the game environment. JADE EMPIRE'S plot consists of a branching story line and many unrelated optional side-quests to surround it. The branching story line depends on the PC's choices throughout the game; one branch leads towards a *good* character ending while another branch leads towards an *evil* character ending. This branching story line resulted in much of the content in JADE EMPIRE having to be doubled.

In the not-so-distant past computer games were authored by a small team of programmers and consisted of simple graphics, simple sounds, and a simple story. Just as the power of computers is increasing exponentially, so is the quality that goes into producing computer games. Currently teams of hundreds of people consisting of programmers, game designers, artists, sound engineers, actors, quality assurance specialists, and others are authoring computer games. These games utilize advanced graphics and sound to give the player a more immersive experience. However this improvement in technology increases consumer expectations for a more immersive and entertaining story.

This presents a challenging problem for computer game authors since an immersive and entertaining interactive story is difficult to create. Not only must the game designers be technically creative, they must be capable of drafting better characters, plots, dialogue, action, and interaction – in short, better stories. Designers and programmers are constantly trying to stay one step ahead by using cutting-edge technology to create engaging stories.

1.1 Authoring Computer Games

Computer games are often designed using a Computer-Aided Design (CAD) tool. A CAD tool allows the designer to visualize and manipulate the game environment as the player will see it instead of using a cryptic low-level textual representation of the player's surroundings. Figure 1.3 shows a screen shot from the Aurora Toolset, a CAD tool used to design game adventures for the game NEVERWINTER NIGHTS [16] (NWN) by Bioware Corp..



```
Loading Tiles
Area verified successfully.
The area Area 001 was successfully saved
```

Figure 1.3: Screenshot of the NEVERWINTER NIGHTS Aurora Toolset, by Bioware Corp.

With the Aurora Toolset a game designer may create a game environment as the player will see it. By placing objects such as creatures, doors, and inanimate objects (called placeables) in the various areas of the game, the game designer knows exactly how those objects will look in the game. Manipulating effects such as lighting and sound allows the game designer to immediately see and hear the changes they have make using the CAD tool.

Use of CAD tools greatly increases productivity as the game designer instantly sees any changes made to the environment. The designer does not have to work in one format to edit a map and then switch to another format to see how their editing changes the game. The designer also edits the environment in an intuitive way, such as painting the terrain or dragging, dropping, and rotating objects using the CAD tool. This is considered the state of the art in designing game settings.

A CAD tool only helps the designer create the setting (environment). A designer cannot use it to design how a player will interact with a game environment – scripting is necessary. Scripting can specify how the game environment will respond to events that happen in the game. Examples of these events include: the death of a creature, reaching a specific point in a conversation, and removing an item from a container. Each event may associate a script with it. A script is a set of instructions telling the game environment how to change in response to the event. When removing an item from a container such as a sarcophagus, a designer might attach a script to create, for example, a mummy creature near the sarcophagus along with a visual effect to highlight the mummy's entrance.

Usually scripts are written in a high-level programming language designed to abstract the process of writing code by avoiding the low-level details found in languages such as C/C++. However, scripting is difficult because scripts require a precise specification; it is programming and many game designers are not programmers. They specialize in designing games not writing code. A script can also be written by a programmer after a designer specifies the intent of the script. This is a tedious task for the programmer whose resources could be spent on other tasks, such as creating new and better tools, adding artificial intelligence to the game and improving the game engine. In addition, information may also be lost while communicating the intent of the script, resulting in both the designer and the programmer having to revisit the script.

Specifying plot in computer games uses scripts. As the plot advances scripts can record and control the game state. Often the plot in CRPGs is divided up into short adventures called quests. An example quest is when the PC is asked to perform a task that involves the killing of a dragon. This quest has three components. First, an NPC asks the PC to kill a dragon. Second, the PC kills the dragon. Third, after killing the dragon, the PC then reports back to the NPC again, who rewards the PC with gold for completing the quest. The following needs to be specified by scripts:

- The NPC dialogue that initiates this mini-plot, where the NPC urges the PC to kill the dragon and updates the PC's journal[1] to remind the PC they are on a quest to kill the dragon.
- When the dragon dies, so as to change the game state and update the PC's journal to remind the PC they have killed the dragon.

[1] A CRPG will often have a journal that records the PC's actions to remind them what they have done. The journal may also give hints as to what the PC has to do next.

- The NPC dialogue that shows after the PC defeats the dragon, where the NPC congratulates the PC on defeating the dragon, gives the PC some gold, and updates the PC's journal to remind the PC they have finished the quest.

- The final NPC dialogue that shows if the PC returns to the NPC after the reward has been given.

Figure 1.4 shows some of the NWN scripts required for this simple quest. A game designer will create and maintain each of these scripts in different locations throughout the CAD tool. A conversation in NWN consists of alternating responses between the NPC and the PC. Scripts can be assigned to these responses to determine when to display the response and what happens when the response is displayed. Scripts 1.4(a), 1.4(c) and 1.4(d) all belong to different responses in the NPC's conversation. However a conversation can be rather large, having thousands of different responses to place scripts. Script 1.4(b) is for the event when the dragon dies. As the number of quests in a game increases it becomes more difficult to keep track of which scripts belong to which quest, making quests difficult to update and maintain. In commercial CRPGs, it is not uncommon to need thousands of scripts to control the plot and provide interactions between the PC and environment. For example, the official campaign story for NWN[2] has 7,857 scripts resulting in 141,267 lines of code.

The dragon quest uses the variables named *DRAGON_DEAD* and *GOLD_GIVEN* to maintain game state; the former is a flag that maintains whether or not the PC has killed the dragon and the latter is a flag that maintains whether or not the PC has been given gold by the NPC. Making scripting calls to the journal interface updates the PC's journal. A game designer uses a journal editor in the Aurora Toolset to create all of the journal entries for the quest. A user references the journal entries using a unique identifier, or tag, of the quest called *Dragon_Quest*. Each script communicates with the other scripts in the quest through the game state variables they query and set. If any one of the scripts misspell one of the variable names, that script will no longer communicate properly with other scripts in the quest. These types of errors are common and difficult to identify, especially as the quests become more complicated, using many variables to communicate the game state between scripts.

Often similar plots will require similar scripts. One example is to repeat the killing-the-dragon example plot with a different NPC who asks to kill an evil knight instead. This evil knight plot can use the same set of scripts shown in Figure 1.4, with minor modifications to variable and tag names as well as changing the locations of the scripts. Unfortunately, this process of copying and replacing variable and tag strings is dangerous, since forgetting to make a change or misspelling a string produces a semantic error in the quest.

[2]NWN is designed so that users can make their own custom adventures to share and distribute. The game comes with set of pre-built adventures that combine to make an epic story line. These adventures are called the official campaign for NWN.

```
int StartingConditional()
{
  object player = GetPCSpeaker();
  int dragonDead = GetLocalInt(player, "DRAGON_DEAD");

  if(! dragonDead)
    AddJournalQuestEntry("Dragon_Quest", 1, player);

  return ! dragonDead;
}
```

(a) Script to check that the dragon has not been killed.

```
void main()
{
  object player = GetLastKiller();

  SetLocalInt(player, "DRAGON_DEAD", TRUE);
  AddJournalQuestEntry("Dragon_Quest", 2, player);
}
```

(b) Script that updates the game state when the dragon dies.

```
int StartingConditional()
{
  object player = GetPCSpeaker();
  int dragonDead = GetLocalInt(player, "DRAGON_DEAD");
  int goldGiven = GetLocalInt(player, "GOLD_GIVEN");

  if(dragonDead && ! goldGiven)
  {
    AddJournalQuestEntry("Dragon_Quest", 3, player);
    GiveGoldToCreature(player, 100);
    SetLocalInt(player, "GOLD_GIVEN", TRUE);
  }

  return dragonDead && ! goldGiven;
}
```

(c) Script to check that the dragon is killed and the money has not been given.

```
int StartingConditional()
{
  object player = GetPCSpeaker();
  int goldGiven = GetLocalInt(player, "GOLD_GIVEN");

  return goldGiven;
}
```

(d) Script to check that the money has been given.

Figure 1.4: The set of scripts needed for a quest that involves the slaying of a dragon.

1.2 Patterns in Narrative

The previous section shows two different plots that require similar sets of scripts. Both plots include a pattern where an NPC requests the PC to kill another NPC. The idea of patterns in narrative is quite common.

A good example of patterns in fiction is the work of Vladimir Propp [20]. Propp identifies several types of patterns that occur in Russian folk-tales. In these patterns different characters in the folk-tales are identified through roles such as the hero, the victim, the villain, etc. The roles then interact with one another in several different types of patterns. An example of a pattern is a hero returning home to discover that the victim has been kidnapped by the villain. These patterns also apply to other forms of media, such as novels or movies, where patterns such as love-triangles and closed-room mysteries abound.

Patterns also exist in computer role-playing games. Often the PC is asked to go on a quest to advance the plot. While there are many different ways for an author to write a quest, specific quests can be grouped together into a small number of patterns. The kill-the-dragon quest example can be subsumed by a *kill a creature* quest pattern. Another common quest pattern is the *retrieve/deliver an item* quest. In this type of quest, an NPC asks the PC to find an item and return it to the NPC or to deliver it to another NPC. There are many different ways that the PC can acquire the item, such as having to persuade another NPC to give up the item or perhaps having to kill a monster who has it or is guarding it. The motivation for why the NPC wants the PC to retrieve the item can be whatever the game author chooses. Despite these differences, the intent of all of these quests is the same: *retrieve/deliver an item* for an NPC.

In the official campaign story for NWN, an NPC named Aribeth asks the PC to find four ingredients to create a cure for a plague. This quest is an adapted version of the *retrieve/deliver an item* quest, where instead of retrieving one item the PC must retrieve four items. Section 1.1 shows that each quest may need several different scripts in several different places. It is difficult for game designers to write and keep track of these scripts. For the Aribeth quest there must be scripts for when the PC acquires each of the four items, as well as scripts in Aribeth's conversation to control what she says depending on the state of the quest. Because this quest has scripts in various locations throughout the story, it is difficult to both track and maintain these scripts.

The complex story lines of modern CRPGs require hundreds or thousands of main plot quests and optional quests (called side-quests). These optional side-quests greatly increase the player's sense of freedom within the story. Unfortunately, quests in most commercial CRPGs are simple in nature, since more complex quests require more complex scripts and thus more (costly) programmer time. Game designers need an abstraction mechanism for creating quests. Just as a CAD tool is necessary for game designers to create quality settings, an abstraction mechanism and associated tool is needed for creating quality quests. Such a quest tool would free the game designer from the difficulties associated with quest scripting. The time saved can be used to construct more intricate quests.

The bulk of game development cost is presently dominated by content creation. This requires investing a huge amount of resources to give games high-quality artwork, realistic characters, fully-orchestrated soundtracks, and sweeping story lines. Anything that can be done to improve the content creation process results in less cost to develop a game or better content being developed for the same cost. Currently, scripting is the bottle-neck in creating a good story line, with patterns a solution to making them easier to develop.

1.3 Automatic Story Generation

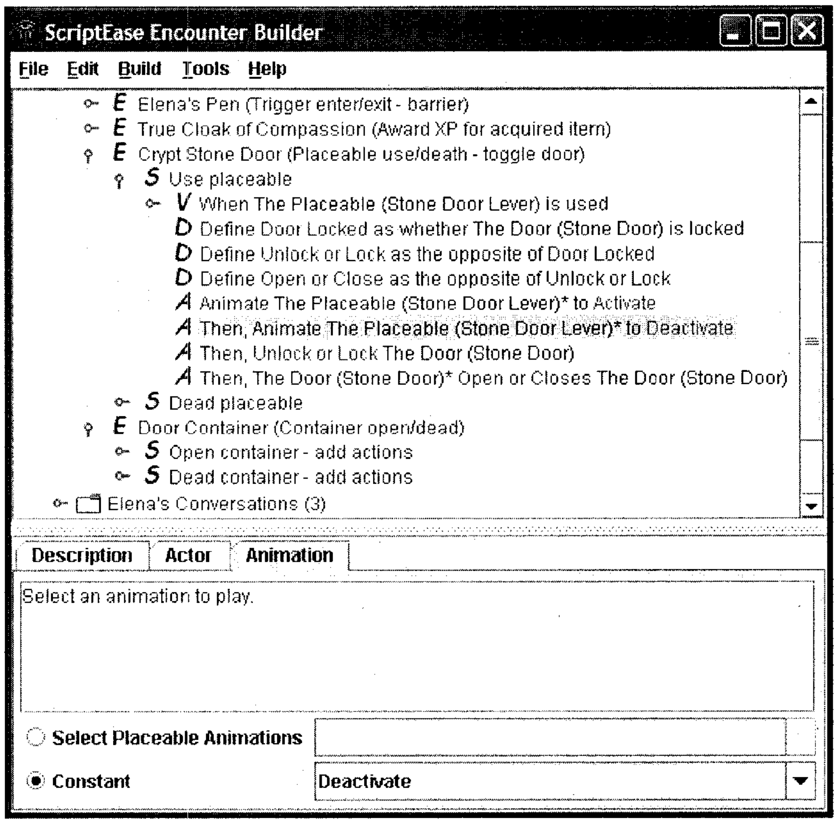
Rather than help the game designer to design quests, there have been attempts to automatically generate stories for the game designer. For example, story directors are one attempt at automatic story generation [21]. A story director tries to maintain a specific story line while keeping track of all the NPCs and the PC. If a PC or NPC acts in such a way that it is impossible to complete the current story, then the story director dynamically alters the story to one that can be completed. Narrative mediation is similar to story directors in that the mediator still tries to maintain a specific story line [22]. With narrative mediation, the mediator has control of everything in the story except the PC. This allows the mediator to react to the PC's actions. The methodology of narrative mediation is similar to planning in that the mediator considers alternative story lines ahead of time, so that when the PC takes an action that deviates from the current story line, the mediator immediately acts upon a new story line that has been planned.

These forms of story generation are dynamic in that the story is created on the go, but they fail to let the game designers maintain control. Just as good linear stories are written by human authors, good interactive stories should remain primarily in the hands of game designers.

1.4 Pattern Abstraction

A quest pattern is a useful abstraction that an author can adapt so that the pattern meets specific story needs. An example quest pattern is the *retrieve/deliver an item* pattern. The game designer need only specify three things to adapt the pattern: a point in a conversation asking the PC to retrieve or deliver the item, the item to retrieve or deliver, and the NPC that should receive the item. For example, in the official campaign for NWN the PC has a conversation where an NPC named Jemanie asks the PC to find his brother. This does not appear to be a *retrieve/deliver an item* quest pattern but it turns out to be one as Jemanie's brother is dead. To prove this, the PC must return a ring found on the brother's body to Jemanie.

By providing tools for game designers to help with plot, the game designers can make either more plots, or more complex plots, or use the savings for other development activities. Quest patterns provide a good reuse mechanism to designers since they contain correct functionality while adaptation allows designers to quickly make more complex plots.


ScriptEase Encounter Builder
File Edit Build Tools Help
Description Actor Animation

```
E Elena's Pen (Trigger enter/exit - barrier)
E True Cloak of Compassion (Award XP for acquired item)
E Crypt Stone Door (Placeable use/death - toggle door)
  S Use placeable
    V When The Placeable (Stone Door Lever) is used
      D Define Door Locked as whether The Door (Stone Door) is locked
      D Define Unlock or Lock as the opposite of Door Locked
      D Define Open or Close as the opposite of Unlock or Lock
      A Animate The Placeable (Stone Door Lever)* to Activate
      A Then, Animate The Placeable (Stone Door Lever)* to Deactivate
      A Then, Unlock or Lock The Door (Stone Door)
      A Then, The Door (Stone Door)* Open or Closes The Door (Stone Door)
  S Dead placeable
E Door Container (Container open/dead)
  S Open container - add actions
  S Dead container - add actions
Elena's Conversations (3)
```

Select an animation to play.

○ Select Placeable Animations

● Constant: Deactivate

Figure 1.5: Screenshot of ScriptEase.

A tool that applies patterns successfully to NWN is ScriptEase [13] (see Figure 1.5). ScriptEase uses generative design patterns to provide the game designer with an alternative to scripting. A designer creates a specific instance of the pattern and specifies the options that differ between pattern instances. The *container disturb - spawn creature* pattern in ScriptEase has three options: a container that will have its inventory disturbed, a creature that spawns when the container is disturbed, and a visual effect to show on the creature that spawns. Once a designer instantiates a pattern and specifies its options, then the designer adapts the pattern instance for a particular story. For example, the designer can adapt the *container disturb - spawn creature* example by having the PC speak something regarding their surprise at the creature that spawns.

The patterns are generative in that through instantiating a pattern and then adapting it, ScriptEase can generate scripting code from the instantiation without the game designer having to write a line of it. This allows the game designer to avoid the problems associated with writing scripts by having them work at the abstract level of patterns. Currently ScriptEase provides *encounter* patterns for the simple interactions between the PC and inanimate objects in the game [13], as well as *behaviour* patterns to control behaviours that characters in the game may exhibit [5].

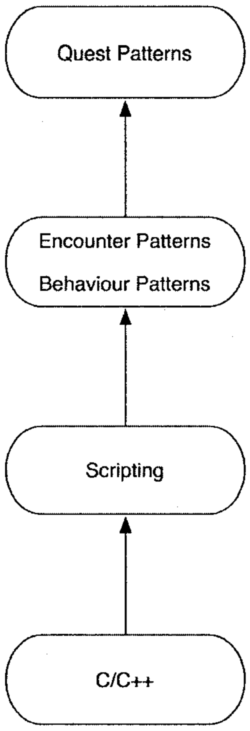


Figure 1.6: A hierarchy of abstractions used for creating the plot in CRPGs.

Patterns are just one level in a hierarchy of abstractions. At the lowest level are programming languages like C/C++. At this level, programmers are expected to deal with details that are not specific to game design, such as reading input from the keyboard or sending graphical output to the screen. Scripting is the next level of abstraction, where all of the details not related to game design are abstracted from the scripter. The scripter now has access to a library of functions that deal directly with controlling the game, but this level still requires the scripter to have knowledge about programming. Game development at the level of patterns no longer requires programming knowledge. Patterns conceptualize game control into a form that is intuitive and easy to use for non-programmers (e.g., high-school students [3]).

The research that this dissertation describes introduces a new level of abstraction for plot development in CRPGs that is higher than *encounter* and *behaviour* patterns. The name for these new patterns is quest patterns. Manually scripting the plot in CRPGs requires multiple scripts to work together. By generating plot scripts from quest patterns, many *sub-patterns* must generate scripts that work together to form a cohesive story. Therefore, quest patterns are meta-patterns that describe how a set of patterns should work together, by providing the user with a high-level quest construct. This eliminates the need to look at each pattern being used in the quest. Figure 1.6 shows the hierarchy of these abstractions, with quest patterns being the highest level of abstraction while C/C++ is the lowest.

1.5 Contributions

This research makes several contributions that are intended to aid game designers in the generation of quest content. This is an area where there is a current lack of deployed tools and active research in game design.

First we introduce the idea of a quest pattern, showing how abstracting a quest at a high-level can make it easier to envision and manipulate the stories in CRPGs. We give an implementation of generative quest patterns in ScriptEase, allowing users to work with and adapt quest patterns to their specific game needs. This research provides a basic catalogue of quest patterns, which can be used in commercial CRPGs to aid game designers in the rapid development of plot.

To further help game designers, this work presents an implementation of a quest generator that automatically generates quest outlines for game adventures using the patterns from the quest catalogue. These quest outlines are specific enough that they can be quickly implemented in ScriptEase. The quest generator is very helpful in the rapid creation of side-quests. Side-quests are simple quests not relevant to the main plot of a game. It is important for CRPGs to have a large number of side-quests as they give the PC more choices by providing the feeling of an open and richer virtual world.

1.6 Outline

This dissertation is organized as follows. Chapter 2 describes related work to both structuring and generating the plot in games and literature. Chapter 3 discusses quest patterns and how to use them to structure the plot in games. Chapter 4 gives an implementation of quest patterns in ScriptEase to allow the automatic generation of scripts. Chapter 5 evaluates the effectiveness of quest patterns by using these patterns to generate the scripts for a sample of quests from the NWN official campaign. Chapter 6 describes a tool that uses quest patterns to automatically generate quest outlines that help a game designer in the rapid development of side-quests. Chapter 7 evaluates the quest generator through a user study. Finally, Chapter 8 presents some conclusions and future work.

Chapter 2

Literature Review

This dissertation develops ideas that game designers may use to rapidly develop plot in CRPGs. This chapter surveys the related work. Since interactive narrative is an extension of linear narrative, we examine work done in both linear and interactive narrative. The tools available for aiding in designing and creating plot are studied. While the use of most tools is in analyzing narrative structure, they can also create narrative. Therefore, we give an overview of the field of narrative generation. Narrative generators allow a new genre for computer games called emergent narrative.

2.1 Tools For Plot Design in Linear Narrative

There is a thorough analysis of linear narrative in the literature, especially its structure of plot. Many patterns occur in plot structure, and creating a story around these structures provides an excellent tool for plot design. We describe several of these structures.

Freytag's dramatic structure is a general way of classifying a plot [7]. Its most common use is in the analysis of Shakespearean plays. There are five stages to the dramatic structure, which often correspond to the five acts in a tragedy or comedy. The stages are:

Exposition - begins the story and introduces characters and setting,

Rising Action - introduces tension and conflict to the story,

Climax - a turning point in the story; the conflict and tension reach their peak,

Falling Action - a resolution to the conflict and tension whether for good or ill, and

Denouement - story ending, things are better or worse than they were before.

These five stages are often represented in an image known as Freytag's Pyramid, as shown in Figure 2.1. This dramatic structure is general enough that it can be applied to many stories. It is also a good structure for plot creation as it provides a simple form for adding tension and then removing it to create a complete story.

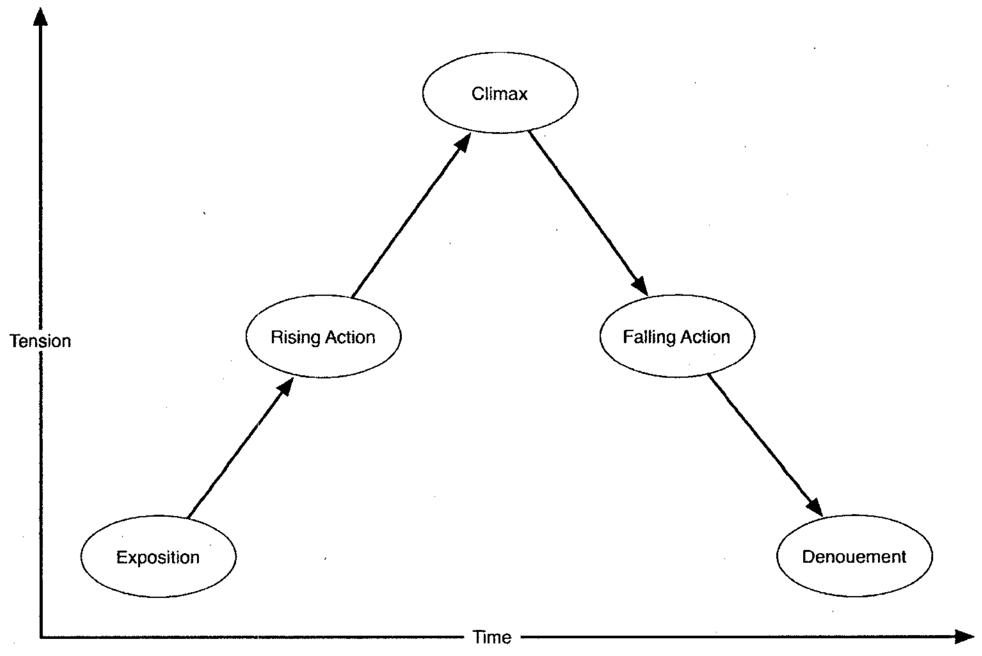


Figure 2.1: Freytag's Pyramid.

Rather than applying a general structure, Vladimir Propp's work identifies a set of functions, or patterns, common to many Russian folk-tales [20]. In these patterns, different characters in the folk-tales are identified through roles such as the hero, the heroine, the villain, etc. We can represent a folk-tale as a sequence of these patterns, or as groups of patterns called moves. An example of a move may be for the hero to return to town to discover that the villain has kidnapped the heroine. These patterns are general enough that they can also apply to other forms of media, such as novels or movies.

Joseph Campbell's work shows a structure common to many myths, called the Monomyth [2]. The Monomyth consists of three parts: departure, initiation and return. Each part has several stages that describe the journey that the hero of the story undertakes. Figure 2.2 shows the various stages in the Monomyth. While Campbell applies the Monomyth structure to several myths, it has also been used to design pieces of narrative, most notably the Star Wars movies by George Lucas [23, 24, 25].

All of this work only helps to visualize the structure of a plot. It does not aid an author in creating an interesting plot, only a well structured one.

2.2 Tools for Plot Design in Interactive Narrative

While many structures have been discovered for linear narrative, the same does not hold for interactive narrative. The addition of player interactivity increases the complexity of the narrative making

- Departure
  - The Call to Adventure - some event occurs that forces the hero to begin his journey (e.g., bandits attack the hero's village).
  - Refusal of the Call - at first the hero may be hesitant to start the journey.
  - Supernatural Aid - a supernatural force will appear and offer to help the hero in their journey.
  - The Crossing of the First Threshold - before the hero can adventure out into the world he is first challenged in their local setting. This threshold symbolizes death.
  - The Belly of the Whale - the hero is thrust into the outside world which is overwhelming and alien to him.
- Initiation
  - The Road of Trials - the hero faces a number of trials to grow and mature.
  - The Meeting with the Goddess - the hero meets a beautiful goddess-like woman, who helps him become whole.
  - Temptation From the True Path - the hero faces temptation from his goal and must prove his worthiness.
  - Atonement with the Father - the hero meets and reconciles with an authoritative figure.
  - Apotheosis - a life changing event occurs, that often involves the hero acquiring much greater powers.
  - The Ultimate Boon - with the hero's new-found powers and having atoned with his father-like figure, the hero is able to perform a deed that benefits all.
- Return
  - Refusal of the Return - the hero often becomes reluctant to return home.
  - The Magic Flight - the hero must make a quick but extraordinary journey to return home.
  - Rescue From Without - external forces rescue the hero.
  - Crossing of the Return Threshold - before returning home, the hero must overcome another challenge. This threshold symbolizes rebirth.
  - Master of Two Worlds - the hero becomes free to travel to and from his local and external worlds.
  - Freedom to Live - the hero completes his adventure allowing everyone to live their lives free and happy.

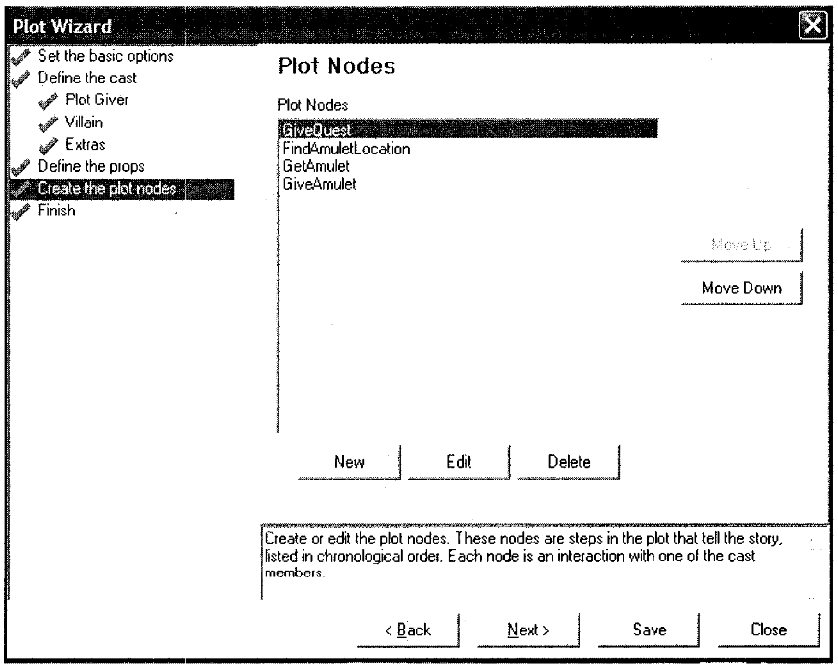
Figure 2.2: The Stages of a Monomyth.

it more difficult to analyze.

One way to handle interactive complexity is to limit the interactivity of the story. This is done in many games. For example, the player may be presented with a short sequence of game play (interactive). After completing the game play the story unfolds through a cut scene which the player has no control over (linear). The process repeats through several sequences of game play, with cut scenes between each sequence. This allows the author to design the story in the same way that they design a linear narrative. The story is interspersed with interactive game play that does not affect the linear story. The disadvantage of this design methodology is that the player has no choice as to how the story unfolds, removing a sense of immersion from their experience. While this may work for certain genres of games, it does not work for genres where the story is important, such as CRPGs.

With computer games, scripting controls the plot, as discussed in Chapter 1. Scripting is not a tool that was designed to specifically control the plot, but it is powerful enough to handle plot. Scripting does not aid plot design, but at least scripting supports the control of more complicated plots.

There is little research related specifically to the design of plot in interactive narrative. This is probably due to the observation that computer games are one of the only mediums in which interactive narrative is used, and computer games are relatively new. Most tools to aid in plot design for computer games are made by game companies for internal use. Since game companies guard their tools, there is little collaboration or communication about plot tools.



Create or edit the plot nodes. These nodes are steps in the plot that tell the story, listed in chronological order. Each node is an interaction with one of the cast members.

Figure 2.3: Plot wizard for the NWN Aurora Toolset, being used to create plot nodes.

One of the few exceptions to this policy is the plot wizard tool for the Aurora Toolset for NWN [17]. It allows a user to create some plots without any scripting. It does this by asking the user to answer several questions, such as which characters and objects are a part of the plot. The plot wizard creates these characters and objects for the user. The wizard also relies on the creation of several small steps in the plot called plot nodes. A plot node corresponds to some action or event that the player may take in the game. Figure 2.3 shows the plot wizard being used to create plot nodes. The plot wizard supports a limited number of events but they cover many of the common events associated with plots. Specifically these events are talking to a creature, talking to a creature to obtain an item, talking to a creature to give an item, killing a creature, killing a creature and obtaining an item, opening a container, obtaining an item from a container, giving an item to a container and opening a door. Each plot node can depend on a previous plot node, allowing nodes to be chained together. This allows a designer to impose an order on which actions the PC must take during the plot. The plot wizard then generates the necessary scripts. If a designer wishes to do something else, they must edit these scripts by hand.

While the NWN Plot Wizard sounds impressive, it has three major limitations. First, the number of events supported for plot nodes is both limited and not extensible. If an author wants to use a more complicated event, such as having the PC use an item at a specific game location, the plot wizard cannot be used. Second, the plot wizard only orders plot nodes linearly. This restricts the types of plots that are possible to create. For example, a plot where the PC finds two items and gives them to an NPC is impossible to create using just the plot wizard. Third, the author cannot adapt the events to make them more interesting. For example, the author could create a plot node where the PC removes an amulet from a chest. Now the author would like to adapt the event so that when the amulet is removed a visual effect that quickly brightens the entire room with a magical light. The plot wizard does not provide the means for the author to make this adaptation. The author could attempt to edit the script that the plot wizard created for the event. If the author then uses the plot wizard again, the edited script may be changed by the plot wizard and the author may lose their changes.

Kismet is a visual scripting language for the *Unreal 3* game engine by Epic Games [27]. Scripts are represented as graphs, where events are diamonds, specific scripting actions are rectangles, and variables are circles. Each event and action has an *in* and an *out*. Drawing a line from the *out* of an event to an *in* of an action means that the action will occur immediately after the event occurs. Drawing a line from this action to another action means that the second action will occur after the first action has occurred. At the bottom of each action are given the parameters and return values. Variables can be attached to the parameters and return values.

While Kismet makes scripting more visual, it does not remove the complexity associated with scripting and programming. One must still be a programmer to understand how Kismet works; the code can simply be viewed in a graph rather than lexically in a window. Kismet provides general support for scripting but has no special features to support plot.

2.3 Generating Narrative

Research into automatically generating linear narrative stories goes back to 1977 [14]. The focus is on creating a complete narrative from beginning to end. This results in the difficult problem of having to ensure that the narrative has good structure while also remaining interesting.

Interactive narrative generation has more liberties. The complete story does not have to be created at once; it can be dynamic. This allows the generator to improvise the narrative. Most research into interactive narrative generation is dynamic.

Fairclough and Cunningham have created an interactive narrative generator for multi-player games using a Story Director (SD) [6]. An SD is an automated agent that works behind the scenes to create a story by providing direction to the players and characters in the narrative. What makes this work interesting is that the SD uses Propp's Russian folk-tale patterns. With Propp's patterns, the authors create a number of different moves to use in the narrative. Each move has a set of roles associated with it. When the interactive narrative begins, the SD looks at the current state. It will then pick a move which closely matches the state. For example, one criteria could be that there are an appropriate number of characters together for the roles in the move, including a PC in the role of the hero, an NPC to play the villain, and an NPC to play the victim. With the move picked, the SD assigns the move's roles to the appropriate characters and they behave by acting out each of the patterns in the order that composes the move.

The SD system allows authors to create pieces of a narrative in a move. Using different sets of moves can create different types of narrative. An interactive narrative created by Fairclough and Cunningham's SD uses a set of moves based on the first Star Wars movie [23].

The moves in this system are similar to quests. While the moves have good structure, they fail to measure what is interesting about the move. There is no emphasis on identifying what motivates a villain to perform an act of villainy, nor is there much emphasis on generating meaningful dialogue. Because the SD works on-line while the game is being played, there is no chance for a human author to intervene and provide motivation or dialogue.

Cavazza, Charles and Mead have an interactive narrative generator that creates stories similar to situational comedies found on television [4]. The player, in their system, does not have an entity in the game. Instead, the player moves objects or suggests ideas to the characters in the game. While the interaction is limited, it does allow the narrative to change depending on the player's actions. The generator assigns each character in the narrative a goal that they must complete. The system focuses on creating plans for each character to achieve their goal. Because the player or the characters may interact, a current plan may not be achievable, in which case the system must create another plan to achieve that goal.

The generator derives plans through the use of a Hierarchical Task Network (HTN). An HTN is

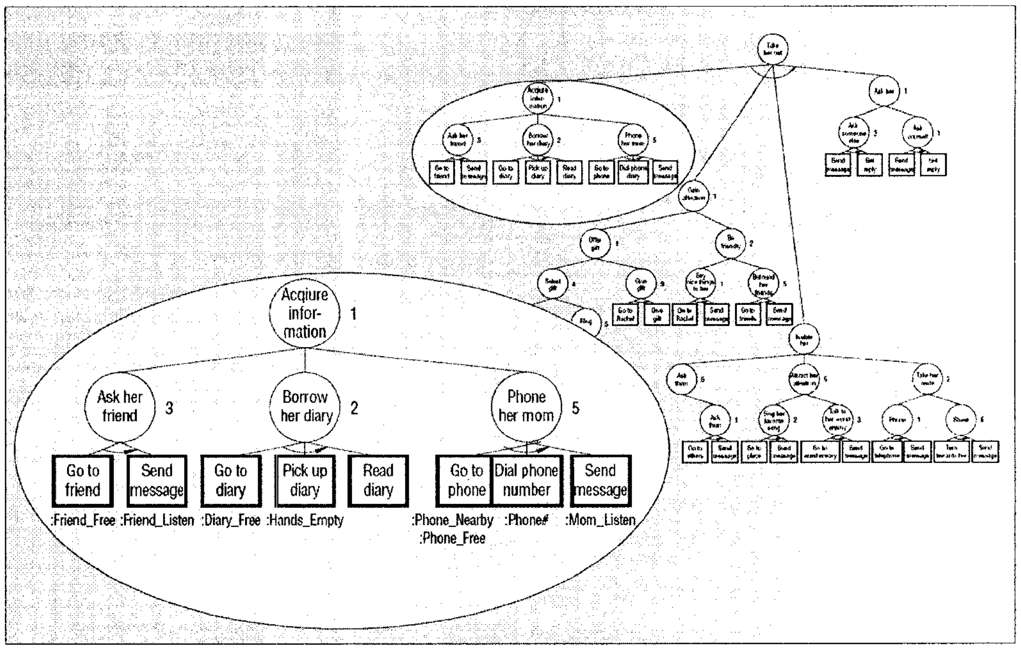


Figure 2.4: HTN Diagram (taken from [4]) to create a plan for a character.

a way of decomposing a goal into a set of tasks required to achieve the goal. These tasks can also be decomposed into sets of sub-tasks. Sometimes completing one sub-task completes the parent task. Other times, completing all of the sub-tasks in the specific order given, completes the parent task. The result is a hierarchy of tasks. Figure 2.4 shows an HTN for a character in the narrative generation system. The task at the top of the HTN represents the character's goal. The tasks in rectangles at the bottom of the HTN are the basic actions that the character can take in the game. Each basic action in the HTN has conditions associated with it. Should one of the conditions fail then the plan fails. Re-planning is done by moving up the HTN from the basic action that failed to a sub-task that is in a list of optional sub-tasks. The generator selects a new optional sub-task and specifies basic actions for the new plan.

The advantage with this system is that it is easy for an author to specify HTNs for the various characters in the narrative. Unfortunately the system does not allow characters to have multiple goals. Once the characters finish their goals, the narrative is complete. This results in an interactive narrative that is short in length, approximately three minutes.

Façade, shown in Figure 2.5, is an immersive narrative that allows the player a large amount of interactivity [12]. A newly-wed couple invites the PC over to their apartment. Upon arriving the player takes part in a drama where it becomes evident that the couple's marriage is in trouble. Because of the high interactivity, there are many resolutions to the drama, such as the player saving or destroying the marriage depending on how they interact with the newly-weds.

Façade uses beats, a collection of behaviours that the NPCs exhibit at different points in the

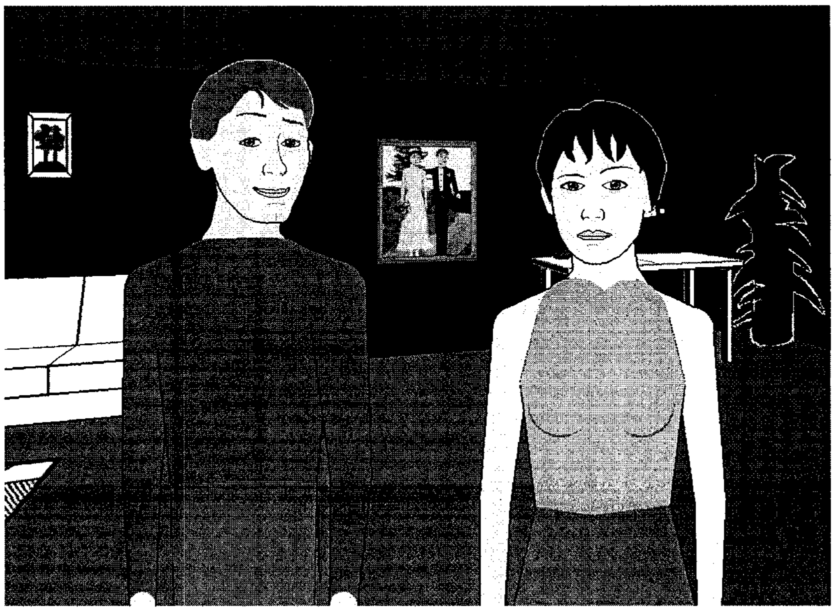


Figure 2.5: Screenshot of Façade.

game, to create the narrative. Different beats occur at different times. When a particular beat is active the NPCs focus their behaviours to move the story in a particular narrative direction. The player's actions can interrupt beats, causing new beats to initiate instead. Each beat has a metric value based on the dramatic tension associated with it. Beats are selected by the level of tension to form a dramatic structure similar to Freytag's Pyramid in Section 2.1. The end result is a sequence of changing beats which creates a dynamic storyline that reacts to the player's actions.

Each behaviour is written by hand, resulting in a time-consuming and story-dependent process. Façade is a game that takes about 20 minutes for one play-through, yet it took about 3 person-years to produce. While Façade successfully creates an interactive narrative, all of the beats were authored for the specific story of a marriage that is falling apart. Certain types of narrative might not be possible in this model, such as those where all of the events are related and come together at the end (e.g. a mystery).

These new forms of narrative generation have created a new genre called the emergent narrative. Here, there is no pre-defined story. The player and the characters in the story all interact with different goals and beliefs and, through interaction, a story emerges. This is interesting as it mimics real life; human history is a long emergent narrative. Emergent narrative suffers from a lack of direction. There is no way to insure that the emerging story will always be interesting for the player. Sometimes an interesting story will emerge but, if human history is any indication, a large number of stories will be rather mundane. In the computer games industry, even having ten percent of your emerging stories being uninteresting can kill a product. A market for emergent games has emerged, but they will not cause the need for games with a pre-defined storyline to disappear.

Chapter 3

Quest Patterns

Game authors will often use quests to make the story modular. Quests segment the large branching plot of a game into smaller independent episodes of story. This makes the game easier for a player to handle, as they can focus on a few quests at a time. A device, such as a journal, is used to remind the player of the status of their quests. Game authors benefit from quests for similar reasons - quests are designed a few at a time. Quests also serve as a way to abstract the plot. For example, the entire sequence of actions needed for the PC to acquire a powerful item can be abstracted as one quest. Taking this further, composing a quest as a sequence of other quests results in a type of composite quest. Main story lines in games are often created from these composite quests.

Many quests found in CRPGs are similar. While the conversations, characters and items used in the quests may be different, the steps the PC must take can be alike. Classifying quests into familiar patterns results in an abstraction that can be reused multiple times. Quest patterns in CRPGs are analogous to design patterns used in software engineering.

3.1 Design Patterns

Software engineering uses design patterns [8] with great success. A design pattern is a general solution to a set of similar software scenarios. To use a design pattern, the user first selects it and then adapts it to fit a specific situation. With traditional (non-generative) design patterns, the user must then write code that implements the design pattern solution. If a generative design pattern is available the code can be generated automatically. If a new set of common scenarios is identified, a new design pattern can be created to address these new scenarios. Once a design pattern exists for a set of problems, the design can be reused. This aids in the rapid implementation of software.

An example of a design pattern is the *Observer Pattern*. The *Observer Pattern* is used when one object needs to monitor the state of another object. The observer object registers itself with the observed object whose state it desires to observe. When the observed object changes state, it notifies all registered observers about this change. The *Observer Pattern* is a common scenario that occurs in GUI programs, such as a spreadsheet editor. There may be several charts and graphs displaying data in regards to a single table. When any of the data in the table is updated, the charts and graphs must update themselves as well.

3.2 Quest Pattern Goals

Quest patterns should provide a high-level mechanism for game authors to structure the plot in CRPGs. They should also be specific enough so that the pattern contains all the details of the quest. A structure for quest patterns is a sequence of actions, or encounters, that the PC may choose to influence the quest. Examples of encounters are the PC reaching a specific point in a conversation, acquiring an item, and killing a creature. Quest patterns must also provide a way for the author to specify what happens in the game in response to changes in the quest state. For example, when a PC kills an evil knight to complete a quest, nearby characters may emerge from their houses since they are no longer threatened by the knight. Other common operations such as giving a journal entry, or awarding experience points[1] (XP) should be handled appropriately.

3.3 Quest Pattern Components

Each quest pattern consists of a set of quest points. Quest points correspond to important encounters that the PC experiences during the quest. The dragon quest mentioned in Section 1.1 has three quest points: first the PC is told about the dragon by the old man, then the PC kills the dragon, and finally the PC reports that the dragon is dead.

At the beginning of the quest the PC can only be told about the dragon by the old man. This quest point is said to be *enabled* while all the others are not. When the PC talks to the old man, the quest point is *reached* and the point is no longer enabled. Reaching a quest point causes other points to enable. In this case the quest point to kill the dragon is now enabled. When the PC kills the dragon the same quest point becomes reached and the final quest point, to report that the dragon is dead, is enabled. This is an example of a linear quest, but not all quests are linear.

Each quest point specifies a label, a type, a list of *enablers* that enable the quest, and a list of encounters that make the point reached. When any of the *enablers* is reached, the quest point becomes enabled. The *enablers* list may also specify that the point is initially enabled at the start of the quest. To reach a quest point, it must be enabled and one of its encounters must occur. Each quest point also specifies a journal entry to add to the PC's journal and the amount of experience to reward the PC when the quest point is reached. Finally the quest point lists additional actions that occur when the quest point is reached. In the dragon quest, when the PC kills the dragon, some additional actions may make the old man continually jump up and down in jubilation over the dragon's death.

[1]CRPGs use experience points to represent the growth (or experience) of a PC. After obtaining a certain number of experience points, the PC will gain a level which increases the powers of the PC. A fighter PC might become stronger and be able to attack more frequently, while a wizard character would become more intelligent and be able to cast more powerful spells.

There are three different types of quest points. When a PC reaches a *normal* point, all points in the quest become disabled except for those points that have this point in their *enablers* list. An *optional* quest point is a point that does not need to be reached to complete the quest. Reaching an *optional* point does not eliminate any previously enabled quest points. The final type of point is a *close* point that completes the quest. When a closed point is reached, all points for the quest become disabled and no new points can become enabled.

Optional quest points provide a way for the PC to take an action without preventing other possible actions from occurring. For example, the quest in Section 1.1 sees the PC killing a dragon at the request of an old man. The quest can be altered so that the PC is initially too weak to defeat the dragon. To gain the strength needed to defeat the dragon, the PC must either acquire an enchanted sword from a nearby sorcerer or drink a potion of strength from an alchemist. Doing either one of these actions gives the PC just enough strength to defeat the dragon. However, if the PC does both of these actions then the dragon can be defeated with little effort. This quest requires the use of optional quest points. After the PC begins the quest by talking to the old man, two quest points are enabled: acquire the enchanted sword and drink the potion of strength. Both of these quest points enable the quest point where the PC kills the dragon. If acquiring the sword and drinking the potion are *normal* quest points, then it becomes impossible for the PC to reach both. This is because when one of the quest points is reached, all quest points become disabled except for those that list the quest point as an enabler. The two quest points do not enable each other, so reaching one point causes the other to become disabled. If both are *optional* quest points, it becomes possible to acquire the sword and drink the potion. This is because reaching an *optional* quest point does not disable any other quest points.

This outline presents the specification for a quest pattern:

- Intent: *The intent of the quest.*
- Options: *Various elements specific to each quest. When a quest pattern is instantiated these options must be set to elements in the game. It is these options that can make multiple instantiations of a quest pattern different.*
- Quest Points
  - *Label of the quest point.*
    - Intent: *The intent of this quest point.*
    - Options:
      - Quest Point Type: *Normal, Optional or Close*
      - Enablers: *The point becomes enabled when any of these points is reached. Alternatively, the quest point may be initially enabled.*
      - Journal Entry: *The entry that is added to the PC's journal when this quest point is reached.*
      - Experience Awarded: *The amount of experience awarded to the PC when this quest point is reached.*
    - Encounters:

        - *The encounters required to occur before the quest point is reached are added here.*
    - When quest point reached:
        - *Additional conditions and actions can be added here for when the quest point is reached.*
  - Additional quest points can be specified similar to the one above.

The example above where the PC must acquire a sword or drink a potion before killing the dragon uses this outline:

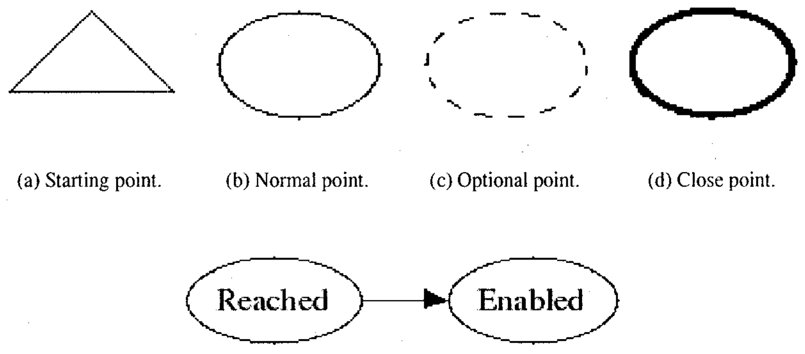
Kill a Dragon Quest

- Intent: *The PC is asked to kill a dragon. Before killing the dragon, the PC must either acquire an enchanted sword or drink a potion to become more powerful. Should both actions occur, the PC will be able to easily defeat the dragon.*
- Options:
  - *Point in old man's conversation where quest is given*
  - *Enchanted Sword*
  - *Potion*
  - *Dragon*
  - *Point in old man's conversation where PC reports dragon is dead*
- Quest Points
  - *Receive Quest From old man*
    - Intent: *The PC talks to the old man to begin the quest.*
    - Options:
      - Quest Point Type: *Normal*
      - Enablers: ⟨*Initially Available*⟩
      - Journal Entry: *An old man has asked me to destroy a nearby dragon. Unfortunately I am not strong enough to defeat the dragon. I'll need to increase my strength. Perhaps I can find a weapon or potion to aid me?*
      - Experience Awarded: 10
    - Encounters:
      - Reaching the *point in old man's conversation where quest is given*
    - When quest point reached:
      - *No additional actions required*
  - *Acquire Enchanted Sword*
    - Intent: *The PC acquires an enchanted sword to aid in defeating the dragon.*
    - Options:
      - Quest Point Type: *Optional*
      - Enablers: *Receive Quest From old man*
      - Journal Entry: *A sorcerer has given me an enchanted sword. This weapon should certainly help me defeat the dragon.*
      - Experience Awarded: 50
    - Encounters:
      - The Player acquires the *Enchanted Sword*
    - When quest point reached:
      - *No additional actions required*

- *Drink Potion of Strength*
  - Intent: *The PC drinks a potion of strength to aid in defeating the dragon.*
  - Options:
    - Quest Point Type: *Optional*
    - Enablers: *Receive Quest From old man*
    - Journal Entry: *An alchemist has given me a potion of strength. After drinking the potion I feel much stronger. This should help me kill that dragon.*
    - Experience Awarded: 50
  - Encounters:
    - The player drinks the *Potion*
  - When quest point reached:
    - *No additional actions required*
- *Kill the Dragon*
  - Intent: *The PC kills the dragon.*
  - Options:
    - Quest Point Type: *Normal*
    - Enablers: *Acquire Enchanted Sword, Drink Potion of Strength*
    - Journal Entry: *The Dragon has been defeated. I should return to the old man and tell him he has nothing left to fear.*
    - Experience Awarded: 100
  - Encounters:
    - The *Dragon* dies
  - When quest point reached:
    - *No additional actions required*
- *Report Dragon's Death to old man*
  - Intent: *The PC tells the old man of the Dragon's death and is rewarded.*
  - Options:
    - Quest Point Type: *Close*
    - Enablers: *Kill the Dragon*
    - Journal Entry: *The old man was happy to hear that the dragon is no more. He has rewarded me well.*
    - Experience Awarded: 500
  - Encounters:
    - Reaching the *Point in old man's conversation where PC reports dragon is dead*
  - When quest point reached:
    - *old man* gives the PC *500* gold pieces

3.4 Quest Pattern Graphs

Quest patterns can also be represented as graphs, giving the author a visualization of the quest. Each quest point has a corresponding graph node that is labeled with the quest point label. In a graph, the starting point serves as a node that describes which quest points are initially enabled. There is no *starting* point in a pattern specification, so one is created automatically in the graph. A *starting* point is represented by a triangle node (Figure 3.1(a)). *Normal* points, are drawn with a solid line (Figure 3.1(b)). *Optional* points, are drawn with a dashed-line (Figure 3.1(c)). *Close* points, are



(a) Starting point. (b) Normal point. (c) Optional point. (d) Close point.

(e) Transition.

Figure 3.1: The different components of a quest graph.

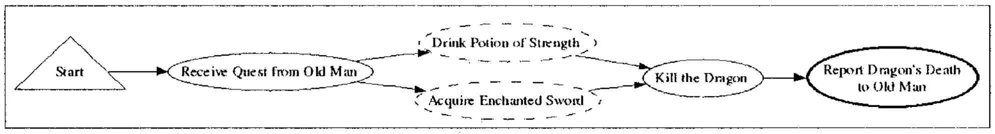


Figure 3.2: Graph of the example quest where the player kills a dragon.

drawn with a bold-line (Figure 3.1(d)). Related points are connected by arrows where the node at the head of the arrow is made enabled when the node at the tail is reached (Figure 3.1(e)).

Quest pattern graphs are a high-level visualization of the quest pattern. They do not display the journal entries, XP awarded, and additional actions associated with each quest point. The quest point labels are often enough information to remind the author what extra information occurs at each point. Implementing these graphs in the proper Graphical User Interface (GUI) would allow a user to point and click at a node in the graph to be presented with this additional information. Given a quest pattern specification, the associated quest pattern graph can be easily and automatically constructed. Figure 3.2 shows the graph for the example quest where the player is asked to kill a dragon.

3.5 Quest Pattern Catalogue

The current quest pattern catalogue consists of five quests: *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 is not a complete catalogue, but it is sufficiently diverse for proof of concept. We are adding more patterns all the time. This section only provides a general description of each pattern, including pattern graphs. A complete specification of each pattern is given in Appendix A.

In the *retrieve/deliver an item* quest, an NPC asks the PC to first acquire an item and then give

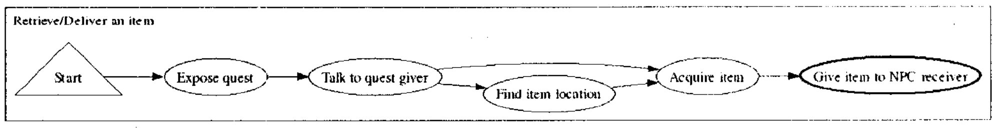


Figure 3.3: Graph of the *retrieve/deliver an item* quest pattern.

the item to an NPC. The reason why the quest is a retrieve/deliver quest is that the PC may have to give the item to the NPC who asked them to retrieve the item, or they may have to deliver the item to a different NPC. Three options must be specified to use this pattern: the conversation point in which the quest is given, the item being retrieved/delivered, and the receiver of the item. If the receiver of the item is the same NPC that the PC conversed with, then the item is being retrieved. Otherwise, the item is being delivered. Figure 3.3 shows a graph of the *retrieve/deliver an item* quest. In addition to the obvious quest points (*talk to quest giver*, *acquire item*, and *give item to NPC receiver*), there are other points (*expose quest* and *find item location*. The other quest points (*expose quest* and *find item location*) are placeholders. The default encounter for these placeholder points is to become reached as soon as they are enabled. The points are there to be adapted by the game author if desired. An example of a *retrieve/deliver an item* quest is where the player finds a missing doll for a young girl. The PC meets a girl who has lost her doll and the conversation includes the *talk to quest giver* point. The PC finds the doll (*acquire item* point). Finally, the PC returns the doll to the girl (*give item to NPC receiver* point). This quest can also be related to the quest where the PC kills a dragon. The PC must help the old man in the dragon quest before his grand daughter (the girl) trusts the PC. The *expose quest* point can be adapted by changing the encounter to be when the Kill a Dragon quest is completed. The *find item location* point can be adapted by having the PC talk to the girl's mother. The mother tells the PC that she last saw her daughter playing with the doll in the market.

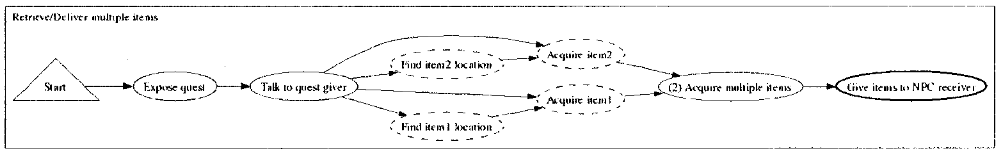


Figure 3.4: Graph of the *retrieve/deliver multiple items* quest pattern.

In the *retrieve/deliver multiple items* quest, an NPC asks the PC to acquire several items and then give the items to an NPC. This pattern is an extension of the *retrieve/deliver an item* quest pattern. The pattern only deals with retrieving/delivering two items but it can easily be modified to handle more items. Four options must be specified to use the this pattern: the conversation point in which the quest is given, the first item being retrieved/delivered, the second item being retrieved/delivered and the receiver of the items. Figure 3.4 shows a graph of the *retrieve/deliver multiple items* quest.

This quest also has points to expose the quest and find the item locations; all quest patterns have these placeholder points. An example of a *retrieve/deliver multiple items* quest pattern is if a mentor NPC tells the PC about four pieces of a broken sword. After obtaining the pieces, the PC gives them to a blacksmith who welds them together to create a new weapon for the PC. The pattern would have to be adapted to use four items instead of two and would also require the closing point to award the PC with the newly created sword.

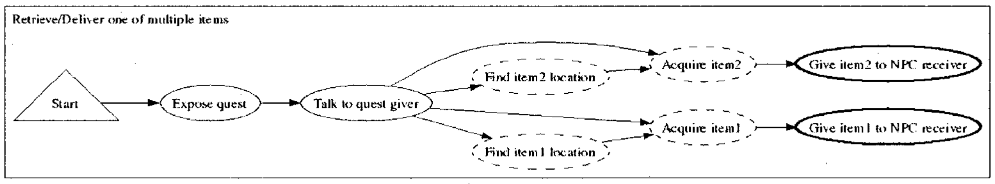


Figure 3.5: Graph of the *retrieve/deliver one of multiple items* quest pattern.

In the *retrieve/deliver one of multiple items* quest, an NPC asks the PC to acquire one of several items and then give that item to an NPC. This pattern is also an extension of the *retrieve/deliver an item* quest pattern. The pattern only deals with retrieving/delivering one of two items but it can easily be modified to handle one of many items. Five options must be specified to use this pattern: the conversation point in which the quest is given, the first item being retrieved/delivered, the second item being retrieved/delivered, the receiver of the first item and the receiver of the second item. Figure 3.5 shows a graph of the *retrieve/deliver one of multiple items* quest. For example, a shop owner may want to buy some items that the PC could acquire, but the shop owner only has enough money to purchase one item from the PC. The *retrieve/deliver one of multiple items* quest pattern can be used for this scenario, where the shop owner is the receiver for both of the items.

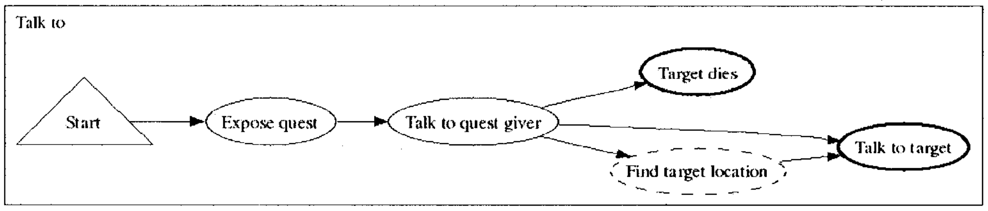


Figure 3.6: Graph of the *talk to* quest pattern.

In the *talk to* quest, an NPC asks the PC to talk to another NPC. However the second NPC could die, which would result in an alternate ending to the quest. While this pattern only handles talking to one other NPC, it can be easily extended to become a chain of talking to many other NPCs. Three options must be specified to use this pattern: the conversation point in which the quest is given, the conversation point in which the PC talks to the second NPC and the other NPC. Figure 3.6 shows a graph of the *talk to* quest. A PC who needs to talk to an NPC may need to talk to several other NPCs to find the location of the necessary NPC. The first NPC they talk to directs them to another NPC. That NPC directs them to another NPC who directs them to the NPC they are looking for. An extended form of the *talk to* quest pattern can be used for this scenario.

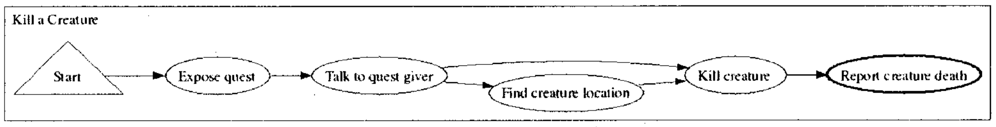


Figure 3.7: Graph of the *kill a creature* quest pattern.

In the *kill a creature* quest, an NPC asks the PC to kill a creature, and then report to an NPC. The report can either be given to the quest giver or another NPC. Three options must be specified to use this pattern: the conversation point in which the quest is given, the creature that is killed, and the second conversation point in which the PC reports that the creature has been killed. Figure 3.7 shows a graph of the *kill a creature* quest. The killing of a dragon example mentioned earlier could use the *kill a creature* quest pattern.

3.6 Using Quest Patterns

Given a quest pattern specification, a programmer could manually write the scripts necessary to implement the quest. The information is specific enough that nothing is lost communicating the pattern instance to the programmer. While no mechanism is given on how to script quests, the rules on how quest points become enabled and reached are detailed enough that a programmer should have no difficulty creating the scripts necessary. A programmer who becomes practiced in scripting quest patterns will be able to quickly create the scripts needed, possibly even reusing some scripts. The scripting process is straightforward enough that the process can be automated. This is shown in the next chapter.

Both the reuse provided by the quest patterns and the decreased scripting time can be used to increase the game content in CRPGs. Not only are quest patterns a novel idea, they are one of the first attempts at providing a formal system for creating plot in CRPGs.

Chapter 4

Quest Patterns in ScriptEase

Quest patterns save time by by applying a formal structure to quest design. This allows the design to be constructed rapidly. Translating a quest pattern specification to scripting code is straightforward enough that it can be done automatically. This chapters describes how ScriptEase automatically generates code for quest patterns.

4.1 ScriptEase Overview

ScriptEase is a tool that uses generative design patterns to create scripts for NWN without a user having to write a single line of code. Users instantiate and adapt design patterns, from which NWScript is generated. All operations in ScriptEase are done at the level of design patterns; the user does not have to write or view any of the scripts.

There are four types of patterns in ScriptEase. Encounter patterns handle interactions with inanimate objects in the game. An example encounter pattern is *container remove – spawn creature*. When a PC removes an item from a container, a monster appears and attacks the PC. Dialogue patterns deal with the conversations between NPCs and the PC. Deciding whether the PC can make a smart or stupid conversation response based on the PC's intelligence statistic is an example dialogue pattern. Behaviour patterns determine how NPCs act in the game. A behaviour pattern may cause an NPC to act as a bartender by taking drink orders, serving customers and fetching supplies from a supply room. Quest patterns, the topic of this thesis, handle quests and how they control the story.

Encounter patterns were the first type of pattern supported by ScriptEase. An encounter pattern has a number of options associated with it, and is composed of one or more scripting scenarios called situations. For example, the *container remove – spawn creature* encounter has a single situation: *remove item – spawn creature*. A more complex encounter, *container disturb - spawn creature*, has two situations: *remove item – spawn creature* and *add item – spawn creature*. Each situation has an event (e.g., remove an item) and one or more actions (e.g., spawn a creature). Each encounter also has options. For example, the *container remove – spawn creature* encounter has two options: the container and the kind of creature to spawn.

A situation also contains definitions and conditions. For example, the *container remove – (specific item) spawn creature* encounter has a definition for the item being removed and a condition that compares this item to a specific option item. The creature is spawned only when the condition is true. Events, definitions, conditions and actions are collectively called atoms. An event atom determines when a situation occurs and is always the first atom in a situation. Removing an item from the inventory of a chest is an example of an event atom. A definition atom defines properties regarding the current state of the game. An example definition atom would be to define whether a NPC has a specific item in their possession. A condition atom tests the results of boolean definitions. An example of a condition atom is a test as to whether an NPC has a specific item in their possession. The action atoms in a situation are only allowed to execute if the test for the condition is positive. Each action atom has a corresponding action in the game. Giving an item to a NPC, is an example action atom.

After the patterns are instantiated a user asks ScriptEase to generate the NWScripts associated with each pattern. The components of each encounter generate appropriate NWN script code. These code fragments are joined together to make a complete set of scripting code for the situation.

Figure 4.1 shows both an instance of the *placeable use - (item not equipped) spawn creature* encounter pattern and the script it generates. When the PC uses an obelisk object and is not wearing an amulet of fire resistance around their neck, then a boy NPC is spawned near the obelisk. Because ScriptEase generates verbose scripts, the script in Figure 4.1(b) has been edited for brevity. The definition, condition, and action atoms in Figure 4.1(a) are labeled *i-iv*. The code snippets associated with each of these atoms are also respectively labeled *i-iv* in Figure 4.1(b).

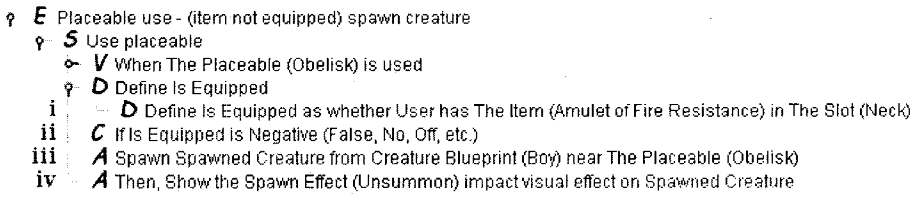
It is expected that most users will be able to build engaging stories using the patterns and atoms already in ScriptEase. However, an advanced user of ScriptEase can also design their own atoms. The user supplies the snippet of *NWScript* associated with the atom. This allows the library of atoms found in ScriptEase to be expanded to include new atoms that do not currently exist.

ScriptEase currently has a catalogue of 70 encounter patterns. A user may also design their own encounter patterns within ScriptEase by either modifying a currently existing pattern or creating a new one from scratch. These new encounter patterns, along with any new atoms, can be grouped into files called *codepaks* which users may share.

4.2 From Quest Pattern Specifications to ScriptEase

The current prototype implementation of quest patterns in ScriptEase makes extensive use of encounter patterns. Future versions of ScriptEase will make quest patterns into first class patterns. This will make quest patterns easier to understand and more efficient. Using encounter patterns to implement quest patterns was a rapid prototyping technique.

The transformation from a quest pattern specification to a pattern in ScriptEase is a straightforward process. At the highest level, ScriptEase represents the quest specification as a new type of



E Placeable use - (item not equipped) spawn creature
- S Use placeable
  - V When The Placeable (Obelisk) is used
  - D Define Is Equipped
    - i D Define Is Equipped as whether User has The Item (Amulet of Fire Resistance) in The Slot (Neck)
  - ii C If Is Equipped is Negative (False, No, Off, etc.)
  - iii A Spawn Spawned Creature from Creature Blueprint (Boy) near The Placeable (Obelisk)
  - iv A Then, Show the Spawn Effect (Unsummon) impact visual effect on Spawned Creature

(a) An instance of a *placeable use - (item not equipped) spawn creature* encounter pattern.

```
void main() {

  // The following are all of the variables used in this situation

  object SpawnedCreature_SE4;
  object AmuletofFireResistance_SE2;
  int IsEquipped_SE3;
  object User_SE1;
  object Obelisk_SE0;

  // This script is attached to the following object's OnUsed script slot
  Obelisk_SE0 = OBJECT_SELF;

  // Define User as the user of Obelisk
  User_SE1 = GetLastUsedBy();

  // Get the object with tag "nw_it_mneck029"
  AmuletofFireResistance_SE2 = GetObjectByTag("nw_it_mneck029");

  // Define Is Equipped as whether User has Amulet of Fire Resistance in Neck
  IsEquipped_SE3 =
    GetItemInSlot(INVENTORY_SLOT_NECK, User_SE1) == AmuletofFireResistance_SE2;

  // If Is Equipped is Negative (False, No, Off, etc.)
  if( IsEquipped_SE3 == FALSE ) {

    // Spawn Spawned Creature from Boy near Obelisk
    location loc = GetLocation(Obelisk_SE0);
    SpawnedCreature_SE4 = CreateObject(OBJECT_TYPE_CREATURE, "malekid003", loc)

    // Show the Unsummon impact visual effect on Spawned Creature
    effect veffect = EffectVisualEffect(VFX_IMP_UNSUMMON);
    ApplyEffectToObject(DURATION_TYPE_INSTANT, veffect, SpawnedCreature_SE4);
  }
}
```

(b) The script generated by the *placeable use - (item not equipped) spawn creature* instance shown above.

Figure 4.1: An instance of the *placeable use - (item not equipped) spawn creature* encounter pattern and the script that it generates.

Option Name	Option Description
Quest	The journal structure in NWN that represents the quest.
Quest Introduction	A text string that is a general introduction of the quest. This introduction is shown above all of the journal entries for the quest. If an empty text string is given then no introduction is shown in the journal.
Intent	A text string to remind the author what the quest is being used for. This option has no function within ScriptEase. It is used only to help ScriptEase authors.
Additional Options	Quest patterns may have additional options based on what the pattern is used for. E.g., a *kill a creature* quest will have the type of creature as one of its additional options.

Table 4.1: The options for a quest pattern in ScriptEase.

Option Name	Option Description
Quest	The journal structure in NWN that represents the quest.
Quest Point Label	A text string that is the label of the quest point.
Quest Point Type	The type of quest point. This can be one of three values: *Normal*, *Optional*, or *Close*.
Enablers	The points that when reached, enables this quest point. This is a text string of the labels of the enabling points, with each label separated by a comma.
New Quest Entry	A text string that is the entry added to the journal when the quest point is reached. If an empty text string is given then no journal entry will be added when the quest point is reached.
XP Awarded	A integer value that is the amount of experience points that the PC is rewarded when the quest point becomes reached.
Intent	A text string to remind the author what the quest point is being used for. This option has no function within ScriptEase. It is used only to help ScriptEase authors.
Additional Options	Quest points may have additional options based on what the point is used for. E.g., a point where the PC acquires an item will have the item as one of its additional options.

Table 4.2: The options for a quest point in ScriptEase.

pattern, the quest pattern. The ScriptEase pattern uses the same options as the quest specification, with some additions. Table 4.1 describes these options. Each quest pattern consists of a list of quest point components. Table 4.2 describes the options for a quest point in ScriptEase. Each quest point contains encounter patterns. If any of the encounter patterns occur, the quest point is considered reached. Each quest point also contains an action block that is executed when the point is reached. An action block contains a list of action atoms and definitions. Optional conditions can also be added to the action block to control whether it is executed.

A user can adapt the quest pattern by adding or removing quest points. When adding a quest point, its options must be set. This includes providing a unique name (label) for the quest point. Other options include the set of enabling points and the type of quest point (*normal*, *optional* or *close*).

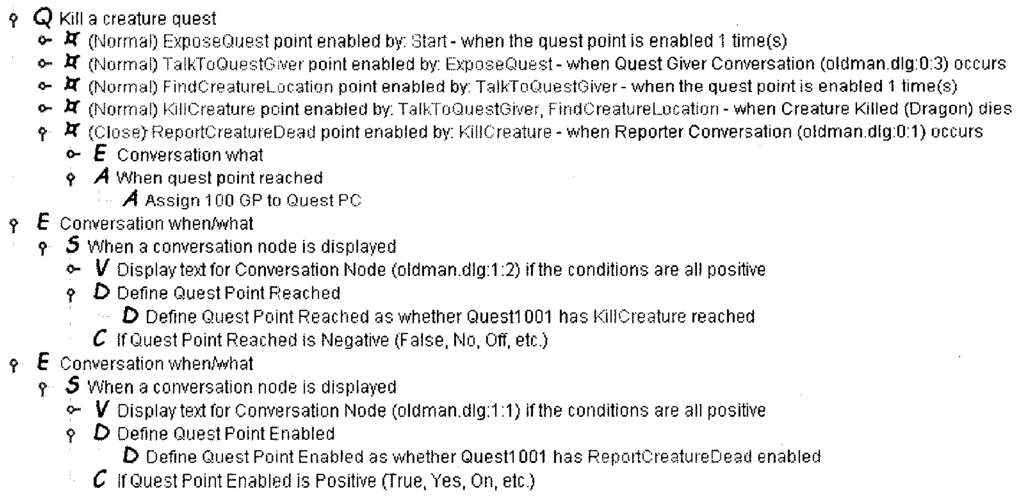
The encounters and action block within a quest point can also be adapted. A user can remove existing encounters, add new encounters, and change the components of the encounters. A common adaption is to add conditions to an encounter to restrict it from occurring. For example, all of the patterns in the quest catalogue have a placeholder quest point that exposes the quest. Without adaptations this expose quest point is reached as soon as it is enabled. A user can add a condition to the encounter to check, for example, if the PC is a ranger[1] to ensure that the quest is restricted to rangers. If the PC happens to be a different class, they never experience this quest.

4.3 ScriptEase Example

Section 1.1 shows the scripting code for a quest where the PC is asked to kill a dragon. Instead of scripting this quest, ScriptEase can be used to generate the necessary scripts using a *kill a creature* quest pattern. Before using the pattern the game author prepares the game adventure. This involves creating the NPC quest giver, the dragon, the conversation that the PC has with the NPC, and the journal quest structure being used.

The author then instantiates a *kill a creature* quest pattern in ScriptEase. The author sets options for the journal quest structure being used, the introduction of the quest, the intent of the quest, the point in the NPC conversation where the quest is given to the PC, the dragon that is killed, and the point in the NPC conversation when the PC reports that the dragon is dead. Further adaptations are possible. For example, in the final quest point where the PC reports that the dragon is dead, the author wants the PC to be rewarded with some gold. This is done by adding an action atom to give gold to the PC in the action block of the final point reached. While not necessary, the author can change the journal entries associated with each of the quest points from the generic entry to specific text relevant to killing a dragon.

[1]In CRPGs the PC will often have a profession or class. Different classes will give the PC different sets of skills to use in the adventure. A ranger class specializes in tracking and hunting skills while a rogue class specializes in sneaking and thievery.



```
Q Kill a creature quest
  ¤ (Normal) ExposeQuest point enabled by: Start - when the quest point is enabled 1 time(s)
  ¤ (Normal) TalkToQuestGiver point enabled by: ExposeQuest - when Quest Giver Conversation (oldman.dlg:0:3) occurs
  ¤ (Normal) FindCreatureLocation point enabled by: TalkToQuestGiver - when the quest point is enabled 1 time(s)
  ¤ (Normal) KillCreature point enabled by: TalkToQuestGiver, FindCreatureLocation - when Creature Killed (Dragon) dies
  ¤ (Close) ReportCreatureDead point enabled by: KillCreature - when Reporter Conversation (oldman.dlg:0:1) occurs
    E Conversation what
    A When quest point reached
      A Assign 100 GP to Quest PC
E Conversation when/what
  S When a conversation node is displayed
    V Display text for Conversation Node (oldman.dlg:1:2) if the conditions are all positive
    D Define Quest Point Reached
      D Define Quest Point Reached as whether Quest1001 has KillCreature reached
    C If Quest Point Reached is Negative (False, No, Off, etc.)
E Conversation when/what
  S When a conversation node is displayed
    V Display text for Conversation Node (oldman.dlg:1:1) if the conditions are all positive
    D Define Quest Point Enabled
      D Define Quest Point Enabled as whether Quest1001 has ReportCreatureDead enabled
    C If Quest Point Enabled is Positive (True, Yes, On, etc.)
```

Figure 4.2: ScriptEase instantiation of a *kill a creature* quest pattern and two encounter patterns.

The instantiated quest pattern does not generate the scripts that control the NPC's conversation. While these scripts are necessary, they are not considered the responsibility of quest patterns. The control of conversations is considered to be in the domain of dialogue patterns. However, quest patterns provide an interface for querying the plot by asking whether or not a quest point is reached or enabled. Since dialogue patterns are currently not implemented in ScriptEase, the author can instead instantiate two encounter patterns to control the NPC's conversation. Each pattern instance determines whether or not to display a point in the NPC conversation by querying the state of the quest. Definitions and conditions for querying quest state have been constructed.

Figure 4.2 shows the instantiated *kill a creature* quest pattern and two encounter patterns, for the example given in Section 1.1. To keep the figure small, only the *ReportCreatureDead* quest point is expanded. This is the only quest point that is adapted, by adding an extra action atom to give the PC 100 gold pieces. The two *conversation when/what* encounter patterns control the flow of the quest giver's conversation. They are not part of the quest pattern itself. The first *conversation when/what* pattern instance checks if the PC has not reached the *KillCreature* quest point, while the other *conversation when/what* pattern instance checks if the PC has the *ReportCreatureDead* quest point enabled.

4.4 Scripting Mechanisms

Quests in NWN are represented as journal structures that are used to add entries to the PC's journal. Each journal structure has a list of journal entries. The journal displays these entries under the heading of the journal structure. Journal structures also have tags to uniquely identify each quest; these tags can be used in scripts.

To script a quest, certain variables must be stored and accessed to maintain the state of the quest.

Since each instance must have a unique journal structure associated with it, the tag of the journal structure serves as an identifier for a quest pattern instance. Appending this tag to other identifiers produces variable names that can be used to store data. For example, if the tag of the journal quest is *DragonQuest* then a possible variable name is *DragonQuest_QuestIsCompleted.* This variable contains a boolean value (*true* or *false*) that states if the quest is completed.

Some variables are initialized at the beginning of the quest. This includes the quest points that are initially enabled. The initializing scripting code is executed when the PC starts the game adventure.

Each quest point has a unique label associated with it. To obtain a scripting identifier for a quest point, the label of the point is appended to the journal structure tag. Appending additional identifiers to this creates variable names that store information about the quest point. For example, a quest with the journal tag *DragonQuest* has a point with the label *KillDragon.* A boolean variable named *DragonQuest_KillDragon_QuestPointIsReached* stores if the quest point is reached.

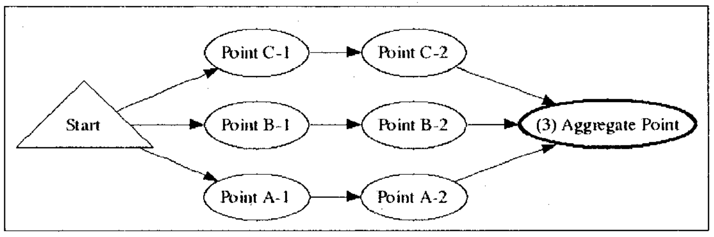
With this naming scheme, any type of data about a quest can be stored in variables. Deciding when to update the state of these variables is a different matter. There are two special events used in ScriptEase quest patterns: an event for when a quest point becomes enabled *N* times and an event for when a quest point becomes reached. NWN only has a specific number of events defined and there is no support for these two special quest events. To solve this problem, the special quest events are emulated by placing all of the code for a quest into one script. Whenever a quest event is supposed to occur, the quest script is executed instead. Each event is treated as a rule. If the conditions for a rule are satisfied then the event occurs and the related code for that event is executed. This allows all of the code needed for a quest to be centralized into one script. The only code for a quest that is located outside of the quest script is found in the encounters associated with each quest point.

4.5 Limitations

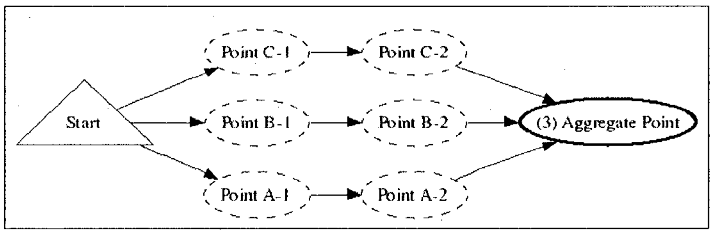
Quest patterns in ScriptEase can be used to rapidly create quests in NWN. While this is useful, there are currently three important limitations to ScriptEase quest patterns.

First, it is inappropriate to have loops in quests because a PC should not be able to repeat encounters that occurred earlier. In ScriptEase, there is nothing to prevent a game author from adapting a quest pattern instance to have a loop. A loop will not function properly in ScriptEase, because the semantics of ScriptEase does not allow a quest point to become enabled after it has been reached. Currently, when a loop is created, ScriptEase does provide a warning. It would be beneficial for ScriptEase to warn the game author that they have created a loop in which previously reached quest points cannot become enabled again.

Second, neither quest patterns nor their implementation in ScriptEase guarantee that the generated quest is logically possible. An author must be aware that quest patterns do not prevent them from creating a quest which cannot be completed. For example, consider a *kill a creature* quest that asks the PC to kill a dragon. The author could inadvertently report the dragon's death to the dragon instead of appropriate NPC. The PC cannot report that the dragon is dead until the dragon is actually dead, at which point the PC can no longer have a conversation with the dragon. ScriptEase does not detect such errors; it is the responsibility of the author.



(a) Incorrect quest using an aggregate quest point.



(b) Correct quest using an aggregate quest point.

Figure 4.3: The incorrect and correct usage of quest points leading up to an aggregate quest point.

The third problem is subtle. Some quest points act as aggregates, by only becoming reached after becoming enabled a certain number of times. Figure 4.3 illustrates the problem that may occur. Points *A2*, *B2* and *C2* all enable the *aggregate point*. The *aggregate point* is only reached if it becomes enabled three times. This requires points *A2*, *B2* and *C2* to all be reached. Because these *enablers* are from different branches, all points along the branches must be made optional (Figure 4.3(b)). Reaching a point on any one of the branches does not disable the points along the other branches. When the nodes along the branches are not optional (Figure 4.3(a)) it is impossible for the *aggregate point* to become enabled three times. This is because when a point along one branch is reached, it will disable the points along all the other branches. This makes it impossible for all of points *A2*, *B2* and *C2* to be reached. If a user is not careful, they may accidentally create the invalid quest shown in Figure 4.3(a) when they attempt to create the correct quest shown in Figure 4.3(b). Currently, ScriptEase does not detect situations like this.

Chapter 5

Case Study

This chapter describes a case study that evaluates the use of quest patterns. A subset of the quests in the official campaign for NWN have been replaced with quest pattern instances. The adaptations needed for each quest are recorded to determine the effort needed to use the quest patterns.

Chapter 3 and Chapter 4 describes how quest patterns can be used to design quests and with a tool, such as ScriptEase, generate the scripts needed for these quests in CRPGs. The next step is to determine how effective quest patterns are. The optimal case would be to instantiate a pattern and only set the options. Sometimes adaptations are needed beyond that of setting options. Quest patterns that often require a large number of adaptations are not effective as the user will spend a large amount of time adapting the pattern. The structure of the pattern might change, requiring the addition or removal of quest points. The encounters of the quest points might change; some encounters might have to be removed while others are added. Actions can also be added to both the encounters and also to the list of additional actions that occurs when a quest point is reached. All these operations are nontrivial, requiring the user to invest more effort into using the patterns.

5.1 Description of Quests

The *Beggar's Nest* area in the official campaign for NWN has a large number of quests. Some of these quests are side-quests that have no relation to the main story. Other quests are related to the main story-line. This section describes all nine quests found within *Beggar's Nest*. This is approximately 7% of the 126 quests found within the official campaign for NWN.

Beggar's Nest is a section of the city of Neverwinter that contains the city's poorest inhabitants. At the point in the game where the PC reaches *Beggar's Nest*, Neverwinter has been struck by a plague and *Beggar's Nest* seems to have been hit the hardest. To make matters worse, the bodies of several of the plague victims seem to be returning to life as undead creatures. In this environment there are nine quests for the PC to resolve. The first eight quests listed are optional, while the last quest listed relates to the main story line and must be completed by the PC:

A Missing Brother Jemanie is worried about his brother, Torin. Torin has been missing for several days now, and Jemanie fears that something has happened to him. Apparently Torin was involved with a strange cult that may have something to do with the undead infestation. The cult has an estate in the northwest corner of the Beggar's Nest. After exploring the area beneath the cultist estate, the PC finds the body of a young man. A ring on the finger of the body identifies him as Torin. The quest ends when the PC gives the ring to Jemanie and tells him of his brother's death.

Find Jemanie Jemanie is a local who needs to be rescued from the undead infestation. The quest ends when the PC either locates Jemanie or Jemanie is killed by the undead.

Find Krestal Krestal is a local who needs to be rescued from the undead infestation. The quest ends when the PC either locates Krestal or Krestal is killed by the undead.

Missing Guard Ergus, a guard captain at the main gate in the Beggar's Nest, has learned that one of his guards, Walters, has gone missing. Ergus does not know whether Walters fell to the undead or not and would like the PC to keep an eye out for him. The PC discovers that Walters is being held captive by the Sword Coast Boys in their warehouse. After being freed by the PC, Walters leaves to make his way back to the main district gate, ending the quest. An alternative ending to the quest occurs if Walters dies before the PC can free him.

Aldo and Hector Aldo and his wife Mattily are guarding a broken-down wagon in the center of the Beggar's Nest. They will not leave until their companion Hector has returned. The PC finds Hector, who is trying to find parts for his master Aldo's wagon. It is doubtful there are any parts left in the looted district. If the PC returns to tell Aldo about Hector, Aldo asks the PC to lead Hector back to him. This quest has three endings. The first ending sees the PC leading Hector back to Aldo and Mattily. All three travelers then abandon the wagon and leave the Beggar's Nest. The second ending results when Hector dies. After the PC reports Hector's death to Aldo and Mattily, both of them abandon the wagon and leave the Beggar's Nest. The final ending occurs if both Aldo and Mattily die. The PC tells Hector about the deaths of Hector's companions. Hector then leaves the Beggar's Nest.

A Strange Cult A local named Jemanie does not know what has caused the undead infestation in the Beggar's Nest, but he is suspicious of a snake cult inhabiting an estate in the northwest corner of the district. Jemanie says that the undead seem to avoid the building, and do not harm the people that come and go from it. Jemanie may just be worried for his brother Torin, who may be involved with them, and has recently gone missing. Torin apparently let it slip that some strange creature had come to power in the group, and shortly thereafter, the infestation began. Jemanie asks the PC to investigate the snake cult. Jemanie also gives the PC a key that Torin may have used to enter the estate. After investigating, the PC discovers that the estate is home to a snake cult, worshipers of some reptilian creature. It is not clear yet what they might have to do with the undead infestation, and they do not seem willing to offer explanations. They attack intruders on sight. The reptilian creature's name is Gulnan, but where it found the power to infiltrate the group is unknown. After the PC thoroughly explores the estate, a lair is discovered beneath. Likely it is from there, in a place called the Warrens of the Damned, that Gulnan is creating and guiding her army of undead. The infestation she has unleashed on the Beggar's Nest must be stopped. Inside the Warrens of the Damned, the PC can find books which reveal information about the snake cult and how Gulnan came to power within it. It seems that Gulnan is a Yuan-Ti, a magical creature with supernatural powers. The PC finishes the quest by destroying Gulnan and acquiring her Yuan-Ti heart as proof of her death. The snake cult will no longer worship her.

Sword Coast Boys A local named Krestal does not know what may have caused the undead infestation, but he does know that something strange has happened to his old gang, the Sword Coast Boys. The leader of the gang, a man named Drawl, has made a deal that has resulted in the gang being turned into undead. Drawl himself is a powerful undead creature as well. The gang uses a warehouse in the north central area of the Beggar's Nest as a base of operations. After investigating the warehouse, the PC finds Walters. Walters was captured and taunted by Drawl because Walters used to be a member, but gave it up to became a guard. Walters tells the PC that Drawl made a deal with a creature named Gulnan to become a powerful undead creature, and had all the other Sword Coast Boys killed to act as his undead minions. How Drawl got in contact with Gulnan, Walters doesn't know. Drawl is destroyed by the PC, and the link between Drawl and Gulnan is severed. With Drawl out of the way, it may now be possible to find this Gulnan creature and put a stop to her plans. After exploring the Sword Coast Boys' warehouse the PC discovers a lair beneath. Likely it is from there, in the Warrens of the Damned, that Gulnan is creating and guiding her army of undead. Inside the Warrens of the Damned the PC can find books which reveal information about the Sword Coast Boys and how Gulnan came to power within it. It seems that Gulnan is a Yuan-Ti, a magical creature with supernatural powers. The PC finishes the quest by destroying Gulnan and acquiring her Yuan-Ti heart as proof of her death. Gulnan will torment the Sword Coast Boys no longer.

A Lost Soul Bertrand is waiting at a shrine in the Beggar's Nest. He hopes that his brother Marcus arrives there soon. Marcus is a wizard that wears red and has certain family heirlooms. When exploring the Beggar's Nest, the PC may find a journal on a distinctive corpse. The corpse was a man who looked to be a wizard that was overwhelmed by a large number of undead. The journal identifies the corpse as Marcus Penhold, and mentions that he was to meet his brother at a shrine in the the district if things got too dangerous during the undead infestation. The PC also finds a staff on Marcus' corpse. Bertrand must then be convinced that his brother is dead. This can be done one of three ways: attempting to convince Bertrand with just words, show Bertrand the journal or show Bertrand the staff. When the PC convinces Bertrand that Marcus is dead, Bertrand asks the PC for Marcus' journal and staff. The quest ends when the PC gives both items to Bertrand.

Undead Infestation The Beggar's Nest is struggling under an infestation of Undead. It is not known if it is related to a plague that has been sweeping through the city or not. The man who knows the most about the undead infestation is Harben Ashensmith. Harben is located in the Shining Serpent Inn, in the south central area of the Beggar's Nest district. Harben asks the PC to explore the Beggar's Nest and any information regarding who has caused the undead infestation should be reported back to Harben Ashensmith or Drake, both found in the Shining Serpent Inn. Both Harben and Drake suggest that the PC seek out Krestal and Jemanie who may have information regarding some strange local events. Talking to either Krestal or Jemanie leads the PC to the Warrens of the Damned. If neither is talked to, the Warren's of the Damned could be discovered by the PC. This area is the lair of Gulnan, the creature that apparently infiltrated and manipulated Beggar's Nest citizens to form an army of the undead. The PC then destroys Gulnan. She was a Yuan-Ti, and one of several creatures imported to be part of a magical attempt to cure the plague. Her heart will need to be brought to Aribeth, a city official who is preparing the plague cure. Harben should also be told about Gulnan's death. The PC completes the quest when the heart is given to Aribeth and Harben is told about Gulnan. This quest, the *A Strange Cult* quest and the *Sword Coast Boys* quest all share events (e.g., acquiring the Yuan-Ti heart). It is possible for the PC to progress in all three quests when these events occur.

5.2 Case Study Description

The hand written code, created by Bioware, for each quest was replaced with a ScriptEase pattern instance. Each pattern instance was adapted so that the functionality of each quest functioned the same as before. The number of adaptations as well as information regarding the scripts replaced was recorded for each quest. These numbers indicate how much work was required by a game author to use these patterns.

Quest points often have encounters that occur during conversations. There are two types of events found in conversations. One event determines what happens when the conversation point is reached and is called *conversation what*. For example, a PC begins a quest by reaching a specific point in an NPC's conversation. This conversation point will have a script associated with the *conversation what* event. The script will update the game state so that the player begins the quest. The other event is to decide when a point in a conversation is displayed. This event is called *conversation when*. For example, an NPC may have a point in its conversation where the NPC congratulates the PC on completing a quest. This conversation point should only be displayed if the PC has finished the quest. A script would be associated with the conversation point's *conversation when* event. The script would check if the PC has finished the quest.

For the conversations in the case study, *conversation what* scripts were replaced and *conversation when* scripts were not replaced. The scripts associated with *conversation what* scripts all correspond to encounters within quest points. These scripts were all automatically replaced by quest pattern instances. The scripts associated with *conversation when* events always query the quest state, they never change the quest state. These scripts are not automatically replaced by quest pattern instances. Replacing these scripts falls into the domain of *dialogue* patterns. As mentioned in Section 4.1, dialogue patterns aid in how a conversation flows. This is done by generating scripts that are placed in the *conversation when* events of a conversation. Because *conversation when* scripts are outside the domain of quest patterns, all of the scripts for these events were not replaced. Replacing these scripts is not difficult, since they only query the state of a quest to determine if a conversation point should be displayed. However, replacing all of the *conversation when* scripts is tedious as there can be many scripts in various locations throughout multiple conversations, all related to one quest. The *conversation when* scripts were replaced in the *A Missing Brother* quest to show that replacement is straightforward.

Quest Name	Pattern	Actions Added	Encounters Added		Quest Points		Scripts Replaced		Lines of Code Replaced
			Unique	Instances	Added	Removed	Unique	Instances	
Find Jemanie	Talk To	0	1	5	0	2	5	8	30
Find Krestal	Talk To	0	1	4	0	2	3	4	10
A Missing Brother	Retrieve/deliver an item	0	1	1	1	2	2	3	10
Missing Guard	Talk To	0	1	5	2	2	1	1	13
Aldo and Hector	Talk To	1	1	5	7	1	4	5	28
A Strange Cult	Retrieve/deliver an item	3	2	4	5	3	9	9	68
Sword Coast Boys	Retrieve/deliver an item	1	2	9	4	3	7	7	59
A Lost Soul	Retrieve/deliver one of multiple items	0	1	4	2	2	3	4	7
Undead Infestation	Retrieve/deliver an item	1	1	7	4	1	3	4	30
Average		0.67	1.22	4.89	2.78	2.00	4.11	5.00	28.33
Total		6	2	44	25	18	37	45	255

Table 5.1: Quest adaptations.

5.3 Quest Statistics

Each of the nine quests listed in Section 5.1 can be represented as an instance of one of only three quest patterns: *talk to* (four instances), *retrieve/deliver an item* (four instances) and *retrieve/deliver one of multiple items* (one instance). However, the number of adaptations needed to produce each of the nine quests varies.

This section describes the quests and the number of adaptations needed. Table 5.1 lists the quest patterns and other statistics. The first column in Table 5.1 gives the name of the quest. The second column specifies which pattern the quest instances were adapted from. The next column gives the number of actions added to each quest. The fourth and fifth columns describe the number of unique encounters and the number of encounter instances added to each quest pattern respectively. The sixth and seventh columns specify the number of quest points added and removed from the quest pattern respectively. The eighth and ninth columns indicate the number of unique scripts and the number of script instances replaced by the quest pattern instance respectively. The final column gives the number of lines of scripting code replaced by the quest pattern instances. The last two rows in Table 5.1 describes the average and total statistics for the quests.

The types of adaptations listed deal with actions, encounters, and quest points. An action can be added to an encounter or it can be added to the actions that occur when a quest point is reached. The number of actions added to the quests is recorded in Table 5.1. A quest point can have more than one encounter associated with it. Therefore, another type of adaptation is to add additional encounters to a quest point. For example, multiple places in a conversation could cause the same quest point to be reached. This scenario requires multiple encounters. For example, three instances of a *when a point in a conversation is reached* encounter and two instances of a *acquire an item* encounter could be used in the same quest pattern instance. Both the unique number of encounters and the total number of encounter instances is given in Table 5.1. The final type of adaptation is to add or remove quest points. This helps change the structure of the quest instance, since quest patterns do not always provide the desired structure. For example, the *A Missing Brother* quest uses a *retrieve/deliver an item* pattern. There were two ways for the PC to acquire the item, so an extra quest point was added. Also, the *expose quest* point in the pattern was removed since the quest can be obtained at any time. Table 5.1 lists both the number of quest points added and removed.

The script information in Table 5.1 includes the number of scripts replaced and the number of lines of code replaced. Sometimes the same script can be used in two different events. For example, a quest could require the player to kill one of three monsters: an orc, a goblin or a harpy. The same script can be used with multiple events for each of the creatures that is killed. The script would set a variable to show that the player has killed one of the three creatures. In this example, there are three instances of one unique script for killing the creatures. Table 5.1 lists both the number of unique scripts and the number of script instances that the quest patterns replace. Each unique script also has code associated with it. A simple script might be composed of a few lines of code, while a complex script will have many more lines of code. The number of lines of code replaced is the final piece of information listed in Table 5.1.

5.4 Quest Particulars

This section describes how each quest was constructed from the appropriate quest pattern.

5.4.1 A Missing Brother

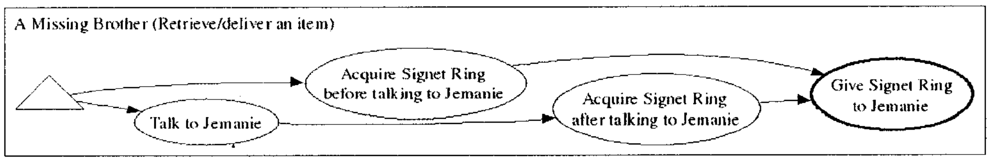

A Missing Brother (Retrieve/deliver an item)
Acquire Signet Ring before talking to Jemanie
Talk to Jemanie
Acquire Signet Ring after talking to Jemanie
Give Signet Ring to Jemanie

Figure 5.1: Quest outline of the *A Missing Brother* quest.

Figure 5.1 shows the quest outline for the *A Missing Brother* quest. This quest has two quest points where the PC can acquire the same item (the Signet Ring found on Torin's body). Normally these two quest points could be combined into one, and this single quest point would be initially available. This would allow the PC to skip the quest point where the PC talks to Jemanie at the beginning of the quest. However in this quest, a different journal entry is given to the PC depending on if the Signet Ring is acquired before or after talking to Jemanie. Since quest points only have one journal entry associated with them, two quest points are needed. The only encounter added to the quest points was an instance of the *when a point in a conversation is reached* encounter.

For this quest the scripts associated with *conversation when* scripts were also replaced. In total six unique scripts were replaced. Each script had one instance found in the *conversation when* events throughout the game. A total of 29 lines of code were replaced. Currently *dialogue* patterns are not implemented in ScriptEase. To replace the scripts, a prototype *encounter* pattern which has a situation that occurs during the *conversation when* event was used. Six *encounter* patterns were used to replace the scripts. Each encounter pattern had to have some definitions and conditions added to them to query and test the state of the quest. In total, eleven definitions and eleven conditions were added to the encounter patterns. No other adaptations were needed. In the rest of the quest, the *conversation when* scripts could have been replaced by encounter patterns, but were not.

5.4.2 Find Jemanie

This quest is one of the simplest. Figure 5.2 shows the outline of the *Find Jemanie* quest instance. No quest points were added and only two were removed from the original *talk to* pattern. A total of five instances of the *when a point in a conversation is reached* were added to the quest points.

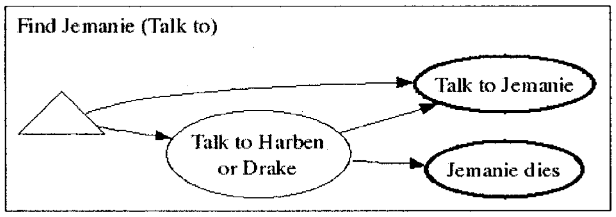


Figure 5.2: Quest outline of the *Find Jemanie* quest.

5.4.3 Find Krestal

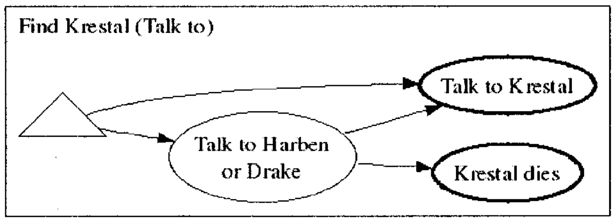


Figure 5.3: Quest outline of the *Find Krestal* quest.

Figure 5.3 shows the outline for the *Find Krestal* quest. This quest is nearly identical to the *Find Jemanie* quest (see Figure 5.2), except that four instances of the *when a point in a conversation is reached* encounter were added to the quest points.

5.4.4 Missing Guard

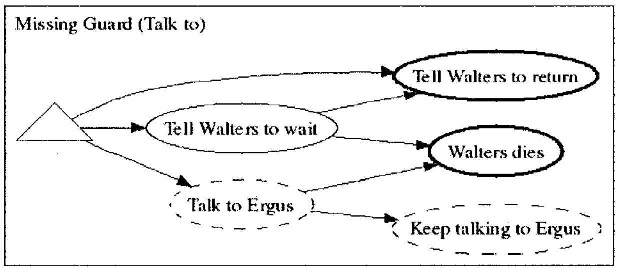


Figure 5.4: Quest outline of the *Missing Guard* quest.

Figure 5.4 shows the outline for the *Missing Guard* quest. This quest uses a *talk to* pattern and uses two extra quest points to give the PC extra information through Ergus, a member of the local militia. five instances of the *when a point in a conversation is reached* encounter were added to the quest points.

5.4.5 Aldo and Hector

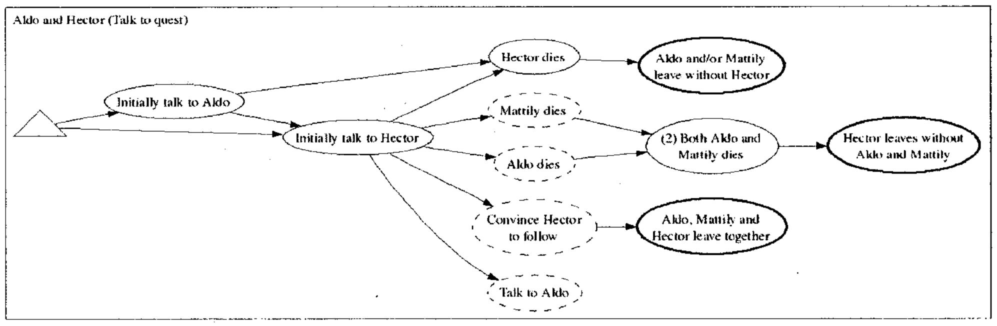

Aldo and Hector (Talk to quest)
Initially talk to Aldo
Initially talk to Hector
Hector dies
Mattily dies
Aldo dies
Convince Hector to follow
Talk to Aldo
Aldo and/or Mattily leave without Hector
(2) Both Aldo and Mattily dies
Aldo, Mattily and Hector leave together
Hector leaves without Aldo and Mattily

Figure 5.5: Quest outline of the *Aldo and Hector* quest.

Figure 5.5 shows the outline for the *Aldo and Hector* quest. This quest has the largest change in structure from its original pattern (a *talk to* quest pattern). A total of seven quest points were added and one quest point was removed.

Seven instances of the *when a point in a conversation is reached* encounter were added to the quest points. This quest also required the addition of a *behaviour* pattern to make Hector follow the PC to Aldo and Mattily. No adaptations other than setting the options were needed for the *behaviour* pattern instance. A single action was added to the quest point where Hector leaves without Aldo and Mattily. This action enabled the *behaviour* to take place where Hector leaves.

5.4.6 A Strange Cult

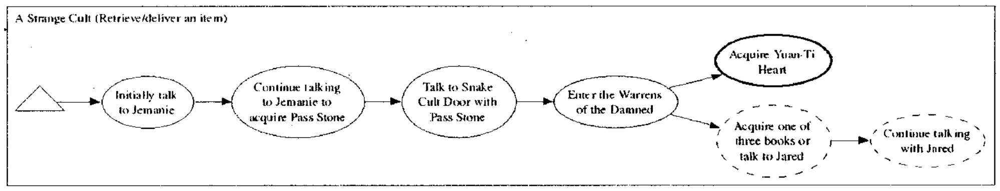

A Strange Cult (Retrieve/deliver an item)
Initially talk to Jemanie
Continue talking to Jemanie to acquire Pass Stone
Talk to Snake Cult Door with Pass Stone
Enter the Warrens of the Damned
Acquire Yuan-Ti Heart
Acquire one of three books or talk to Jared
Continue talking with Jared

Figure 5.6: Quest outline of the *A Strange Cult* quest.

Figure 5.6 shows the outline for the *A Strange Cult* quest. The quest instance is adapted from a *retrieve/deliver an item* pattern. While this pattern only has the PC acquiring one item, this quest instance sees the PC acquiring two items at two different times. The first item is a Pass Stone which allows the PC to enter the cultist estate, and the second item is the Yuan-Ti heart which, when acquired, finishes the quest. Two instances of the *when an item is acquired* encounter were added to the quest points. Two actions were added to the quest points: one was needed for giving the Pass Stone to the PC when talking to Jemanie and the other was needed for destroying the Pass Stone when gaining access to the cult by talking to the Snake Cult Door.

5.4.7 Sword Coast Boys

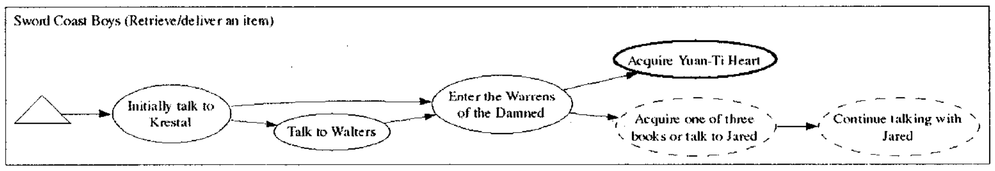

Sword Coast Boys (Retrieve/deliver an item)
Initially talk to Krestal
Talk to Walters
Enter the Warrens of the Damned
Acquire Yuan-Ti Heart
Acquire one of three books or talk to Jared
Continue talking with Jared

Figure 5.7: Quest outline of the *Sword Coast Boys* quest.

Figure 5.7 shows the outline for the *Sword Coast Boys* quest. Starting with the quest point where the PC enters the Warrens of the Damned, the rest of the quest is nearly identical to a portion of the *A Strange Cult* quest. Even the journal entries are similar except that one set of journal entries refers to the cultists and the other refers to the Sword Coast Boys. Seven instances of the *when a point in a conversation is reached* encounter and two instances of the *when an item is acquired* encounter were added to the quest points.

When looking at the human-authored scripts and journal entries created by Bioware for this quest, it became apparent that a large portion of this quest was removed during the creation of NWN. Before the PC enters the Warrens of the Damned through the Sword Coast Boys' warehouse, the PC encounters the Sword Coast Boys' leader, Drawl. Without any dialogue, the PC is attacked by Drawl who has become a powerful undead creature. While this attack has nothing to do with the current quest, the attack might have been related to earlier versions of the quest. The scripts and journal entries seem to indicate that Drawl had a brother named Zelieph. Drawl supposedly killed Zelieph, but the PC might have been able to kill Drawl by summoning Zelieph's ghost to get revenge. None of this side-story ever made it into the final version of NWN. Why it was removed is unknown, but it may be due to the work required to add the extra content to the quest.

5.4.8 A Lost Soul

Figure 5.8 shows the outline for the *A Lost Soul* quest. This quest has two independent paths. The first path sees the PC acquiring the staff and journal and then returning them to Bertrand, which finishes the quest. The second path sees the PC hearing about Bertrand's brother and convincing Bertrand that his brother is dead. Four instances of the *when a point in a conversation is reached* encounter were added to the quest points.

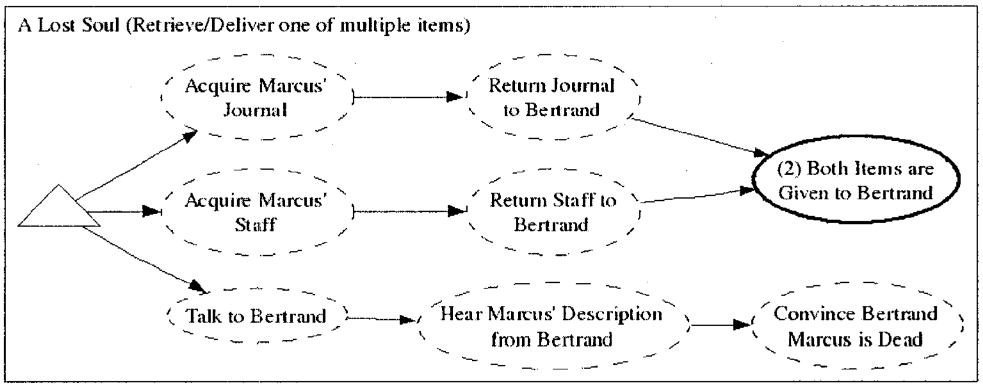

A Lost Soul (Retrieve/Deliver one of multiple items)
Acquire Marcus' Journal
Return Journal to Bertrand
Acquire Marcus' Staff
Return Staff to Bertrand
(2) Both Items are Given to Bertrand
Talk to Bertrand
Hear Marcus' Description from Bertrand
Convince Bertrand Marcus is Dead

Figure 5.8: Quest outline of the *A Lost Soul* quest.

This quest uses an instance of the *retrieve/deliver one of multiple items* pattern. However, the quest requires the PC to bring both the staff and the journal to Bertrand. It would seem that adapting the quest instance from a *retrieve/deliver multiple items* pattern requires less adaptation, but that is wrong. The quest calls for a journal entry to be given when each item is returned to Bertrand and another journal entry for when both items have been given to Bertrand. Because only one journal entry can be given for each quest point, this quest requires a total of three quest points for returning the items. The *retrieve/deliver multiple items* pattern only has one quest point for when the both items are returned. Two more quest points would need to be added to the quest instance, for when each item is returned. The *retrieve/deliver one of multiple items* has a quest point for each of the items returned, but not a quest point for when both items are returned. For returning the items, this pattern only required the addition of one quest point for when both items were returned to Bertrand. This resulted in less adaptations, explaining why *retrieve/deliver one of multiple items* was used.

5.4.9 Undead Infestation

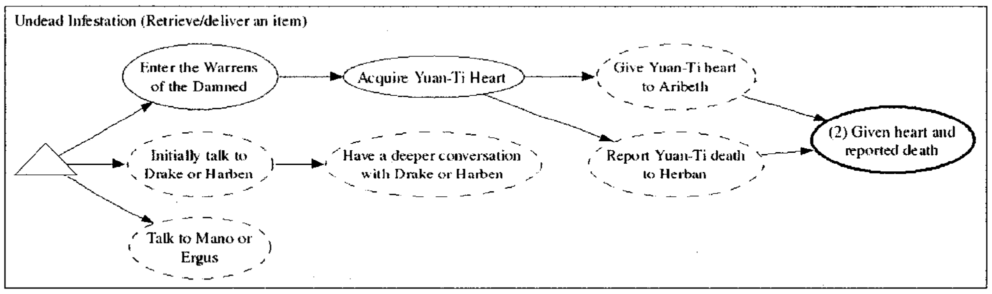

Undead Infestation (Retrieve/deliver an item)
Enter the Warrens of the Damned
Acquire Yuan-Ti Heart
Give Yuan-Ti heart to Aribeth
(2) Given heart and reported death
Initially talk to Drake or Harben
Have a deeper conversation with Drake or Harben
Report Yuan-Ti death to Herban
Talk to Mano or Ergus

Figure 5.9: Quest outline of the *Undead Infestation* quest.

Figure 5.9 shows the outline for the *Undead Infestation* quest, the main quest in the area. The PC is investigating why there are undead in the Beggar's Nest. This investigation leads the PC to the Yuan-Ti Gulnan, whose heart also happens to work as an ingredient in the cure to the plague of Neverwinter. The most obvious way to advance in this quest is to see the player advance in at least one of the *A Strange Cult* or *Sword Coast Boys* quests. Both of those quests see the PC entering the Warrens of the Damned and acquiring the Yuan-Ti heart. The two associated quest points are also points in the *Undead Infestation* quest. Seven instances of the *when a point in a conversation is reached* encounter were added to the quest points. The only action added was to automatically save the game when the PC enters the Warrens of the Damned.

5.5 Discussion

On average, less than a single action was added to each quest. All of the quests required encounters to be added to them. However only two unique types of encounters were used (*when a point in a conversation is reached* and *when an item is acquired*). Whenever encounters were added to a quest point, they were always the same type as the encounter already associated with the quest point. For example, the quest point in the *Undead Infestation* quest (Figure 5.9) labeled *initially talk to Herban or Drake* has several *when a point in a conversation is reached* encounters. Some of the encounters correspond to points in Herban's conversation while the other encounters correspond to points in Drake's conversation. On average, each quest instance has approximately three quest points added and two quest points removed.

Comparing the number of adaptations (actions added, encounter instances added, quest points added and quest points removed) to the number of lines of code replaced is a good way to determine the complexity of using quest patterns versus writing scripts for the quests. Each line of code in a script corresponds to a scripting action. The more scripting actions needed, the more difficult the quest is to script by hand. Similarly, the more adaptations needed for a quest instance, the more difficult it is to use quest patterns. In total, the nine quests required 93 adaptations compared to the 255 lines of scripting code needed to script the quests. Not only were there fewer adaptations than lines of code, but each adaption in ScriptEase generated scripting code that was free of programming errors. Writing scripting code by hand always has the possibility of introducing programming errors. This indicates that using quest patterns is more efficient than writing scripting code. Quest patterns are also easier to keep track of and maintain than writing the scripts for the quests. Scripts, in NWN, are found on various objects and throughout conversations. Finding all of the human-authored scripts for a quest can be frustrating as scripts are located within multiple conversations and objects. With quest patterns, all the information for the quest is stored within the pattern instance. This makes updating the quest easier, as there is no need to track down scripts that are located in several different locations.

Chapter 6

Quest Generation

In CRPGs, the primary purpose of a side-quest is to add breadth to the game by increasing the player's variety of experiences. Since these quests do not need depth, most can be unaltered instances of a quest pattern. Instead of having a game author create hundreds of side-quests for a commercial CRPG, many of these side-quests can be rapidly created through instantiating quest patterns. The options can be automatically picked by an intelligent system.

This chapter describes the Side-QUEst GEnerator (*SQUEGE*), a tool that aids in the rapid development of side-quests in CRPGs by generating quest pattern outlines. *SQUEGE* does not automate the entire process of creating side-quests. Choosing which encounters a PC must perform and creating the scripts for the quests is automated; the former is tedious for game authors when they have to create hundreds of side-quests, while the scripting process is not a focus of their profession. The game authors still must create the conversations and the stories behind the quests. However, these are creative tasks that they enjoy and perform well.

The goal of this work is to generate correct side-quests quickly and conveniently. Given a scene (setting, NPCs, items, containers, etc.), a game author can "push a button" and a side-quest is automatically generated. Quest generation options can be set to bias the generated quests towards simple or complex quests. The author can generate hundreds of different side-quest outlines within minutes, and quickly add the necessary conversations and stories. This guarantees variety and enhances the game experience. In effect, *SQUEGE* can be used to facilitate the rapid generation of important game content, reducing costs and improving the quality of the product.

6.1 Using *SQUEGE*

To show how *SQUEGE* works, we give an example of creating an NWN side-quest. While this example only shows the generation of one simple quest, *SQUEGE* can be used to generate multiple quests of arbitrary complexity within the same CRPG adventure.

First, a game author creates a new game adventure using the NWN Aurora Toolset. The author creates a city setting with several buildings that the PC can enter. The author then prepares the game adventure for quest generation by creating several NPCs, containers, and items that will populate the game. These game objects all have unique names, appearances, and other details. The author places the NPCs and containers throughout various locations inside and outside the buildings of the city. The items' locations will be determined later through the quest generation process.

Because *SQUEGE* is currently implemented as a prototype, making it external to the Aurora Toolset. Hence, the game author must list all the NPCs, containers, and items as input to *SQUEGE*. The author also lists one or more pairs of NPCs located near each other. This allows *SQUEGE* to generate quests in which an NPC pair can have a conversation that the PC may overhear to gain quest clues.

Next, *SQUEGE* automatically generates a side-quest by first randomly selecting a quest pattern from its catalogue of patterns. *SQUEGE* instantiates values for the various options of the quest pattern and produces a graph that acts as an outline for the quest instance. This outline is a graph containing all the information required to create the side-quest.

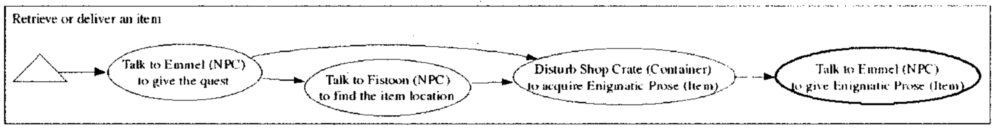
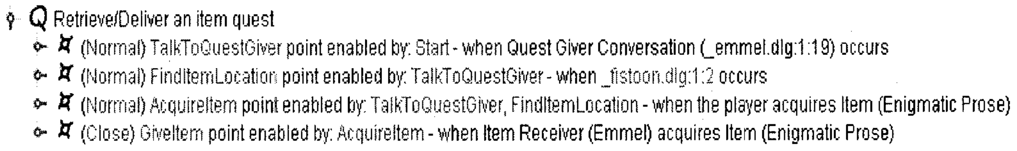

Retrieve or deliver an item
Talk to Emmel (NPC) to give the quest
Talk to Fistoon (NPC) to find the item location
Disturb Shop Crate (Container) to acquire Enigmatic Prose (Item)
Talk to Emmel (NPC) to give Enigmatic Prose (Item)

Figure 6.1: A side-quest outline generated by *SQUEGE*.

Figure 6.1 shows the graph that *SQUEGE* generates for the game author. This *retrieve/deliver an item* side-quest instance requires the PC to perform four encounters: beginning the quest by talking to Emmel, finding the location of the item by talking to Fistoon, acquiring the item in a shop crate container, and giving the item to Emmel in a conversation. The pattern also allows the PC to skip the second quest point (talk to Fistoon to find the item location); the PC can simply stumble upon the item without being told its location. Emmel and Fistoon are NPCs currently in the game adventure. *SQUEGE* does not populate the game adventure with NPCs and other objects, it only uses what the author has placed in the adventure.

The author then decides whether or not to use the side-quest in the game adventure. A side-quest may be rejected for many reasons. For example, the game author may not be able to think of a good story-line for the side-quest outline. In the case of the outline in Figure 6.1, a story-line where the PC retrieves the *Enigmatic Prose* because it contains the names of several assassins is chosen. Should the game author choose to reject the side-quest, a new outline can be quickly generated by *SQUEGE*.

The outline does not provide everything needed for the quest. One of the missing pieces is the text of the conversations for Emmel and Fistoon. The game author creates the side-quest conversations for Emmel and Fistoon in the Aurora Toolset. *SQUEGE* does not attempt to do this, as writing dialogue is one of the game author's specialties. The author may add humour and specific story-related references to make the conversations interesting and unique.



Q Retrieve/Deliver an item quest
- (Normal) TalkToQuestGiver point enabled by: Start - when Quest Giver Conversation (_emmel.dlg:1:19) occurs
- (Normal) FindItemLocation point enabled by: TalkToQuestGiver - when _fistoon.dlg:1:2 occurs
- (Normal) AcquireItem point enabled by: TalkToQuestGiver, FindItemLocation - when the player acquires Item (Enigmatic Prose)
- (Close) GiveItem point enabled by: AcquireItem - when Item Receiver (Emmel) acquires Item (Enigmatic Prose)

Figure 6.2: An instance of a quest pattern in ScriptEase.

The author uses the outline to instantiate the corresponding quest patterns in ScriptEase. The process is straightforward as the author only specifies the options for the pattern. Figure 6.2 shows the pattern instance in ScriptEase. Currently, *SQUEGE* outlines cannot automatically instantiate ScriptEase patterns. However, we are currently engaged in connecting these tools. This will also involve ScriptEase creating placeholder conversations for the game author to alter.

Whether the author must translate side-quest outlines to pattern instances manually or whether the tools are connected, the author still has the opportunity to adapt the side-quest instance in ScriptEase to add a creative touch to the story. One of these adaptations could be to have a bandit NPC appear and attack the PC when the item is acquired. Such adaptations are quick and easy to do in ScriptEase.

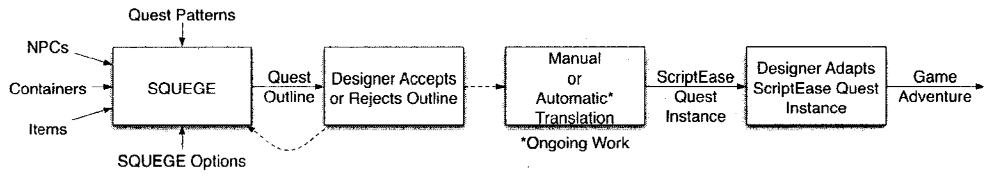

Quest Patterns
NPCs
Containers
Items
SQUEGE
SQUEGE Options
Quest Outline
Designer Accepts or Rejects Outline
Manual or Automatic* Translation
*Ongoing Work
ScriptEase Quest Instance
Designer Adapts ScriptEase Quest Instance
Game Adventure

Figure 6.3: The process *SQUEGE* uses to generate quest outlines.

At this point, the game author has finished generating a side-quest and can now play the adventure in NWN. Figure 6.3 gives a summary of the entire process. First *SQUEGE* uses its pattern catalogue and the input of NPCs, items, and containers to generate an outline of a side-quest instance. Other options are also supplied as input to *SQUEGE* (e.g., the number of side-quests to generate). After outline acceptance and translation, the game author adapts this outline in ScriptEase and generates the required scripts for NWN. If the process is repeated, the author would end up with a completely different side-quest possibly using different NPCs, containers, and items in the same setting.

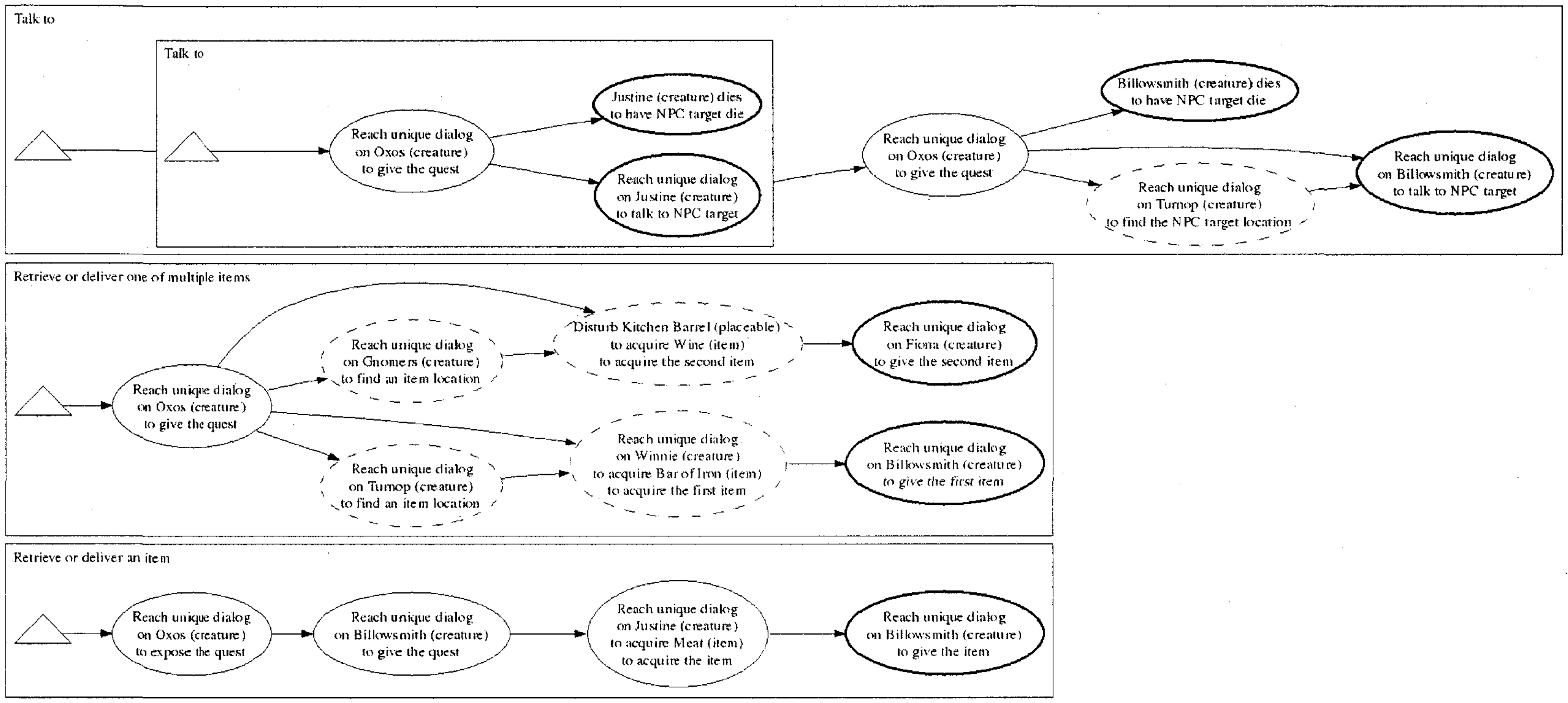

Talk to
Talk to
Reach unique dialog on Oxos (creature) to give the quest
Justine (creature) dies to have NPC target die
Reach unique dialog on Justine (creature) to talk to NPC target
Reach unique dialog on Oxos (creature) to give the quest
Billowsmith (creature) dies to have NPC target die
Reach unique dialog on Turnop (creature) to find the NPC target location
Reach unique dialog on Billowsmith (creature) to talk to NPC target
Retrieve or deliver one of multiple items
Reach unique dialog on Oxos (creature) to give the quest
Reach unique dialog on Gnomers (creature) to find an item location
Disturb Kitchen Barrel (placeable) to acquire Wine (item) to acquire the second item
Reach unique dialog on Fiona (creature) to give the second item
Reach unique dialog on Turnop (creature) to find an item location
Reach unique dialog on Winnie (creature) to acquire Bar of Iron (item) to acquire the first item
Reach unique dialog on Billowsmith (creature) to give the first item
Retrieve or deliver an item
Reach unique dialog on Oxos (creature) to expose the quest
Reach unique dialog on Billowsmith (creature) to give the quest
Reach unique dialog on Justine (creature) to acquire Meat (item) to acquire the item
Reach unique dialog on Billowsmith (creature) to give the item

Figure 6.4: Four side-quests generated by *SQUEGE*.

Figure 6.4 shows four side-quests that were generated by *SQUEGE*. The first side-quest contains another side-quest within it. Completing the inner side-quest is required before beginning the outer. *SQUEGE* has no difficulty in creating four or even 40 side-quests for the same game adventure.

6.2 Technical Details

The previous section illustrated *SQUEGE*'s generation process. *SQUEGE* uses approximately 1,000 lines of Prolog code to generate side-quests in three stages. In the first stage, *SQUEGE* generates a side-quest by selecting a quest pattern from its catalogue of patterns. Each pattern in the catalogue has a weight associated with it. Patterns with a higher weight have a greater chance of being picked. For example, *SQUEGE* can select the *retrieve/deliver an item* pattern. Each quest point in the selected pattern contains a weighted list of possible encounters. In the second stage, one is selected. For example, the quest point where the PC acquires the item has two encounters in the list: the PC acquires the item from a container such as a chest or the PC acquires the item by talking to another NPC. In our example, *SQUEGE* has selected the former encounter. This encounter has two objects associated with it: the container that holds the item and the item itself. In the third stage, *SQUEGE* uses the lists provided by the game author to randomly select appropriate objects. This process occurs for each quest point, so that every point is assigned an encounter and appropriate objects.

The *retrieve/deliver an item* pattern illustrates how a pattern is represented in *SQUEGE*:

Quest: *retrieve/deliver an item*

- Quest Point - ExposeQuest
  - Type - Normal
  - Enablers - ⟨*Initially Available*⟩
  - Encounters - (2) NULL, (1) Overhear conversation between *NPC-Pair*, (1) Talk to *NPC*, (1) Complete a quest
- Quest Point - TalkToQuestGiver
  - Type - Normal
  - Enablers - ExposeQuest
  - Encounters - (1) Talk to *NPC:QuestGiver*
- Quest Point - FindItemLocation
  - Type - Normal
  - Enablers - TalkToQuestGiver
  - Encounters - (2) NULL, (1) Talk to *NPC*

- Quest Point - AcquireItem
  - Type - Normal
  - Enablers - TalkToQuestGiver, FindItemLocation
  - Encounters - (1) Disturb *Container* to acquire *Item:ItemFound*, (1) Talk to *NPC* to acquire *Item:ItemFound*
- Quest Point - DeliverItem
  - Type - Close
  - Enablers - AcquireItem
  - Encounters - (2) Talk to *NPC:QuestGiver* to deliver *Item:ItemFound*, (1) Talk to *NPC* to return *Item:ItemFound*

Each quest point gives a weighted list of possible encounters. The pattern creator sets the weights. Before generating a quest the game author may change the weights if desired. The weights (numbers preceding each encounter) determine the probability of an encounter being picked; higher weights yield a higher probability. Some of the possible encounters are listed as *NULL*. Selecting a *NULL* encounter simply eliminates the quest point. For example, this occurs with the side-quest shown in Figure 6.1. The *NULL* encounter was selected for its *ExposeQuest* point. The point ceased to exist and its enabler (the *starting* point) and point it would enable (Talk to Fistoon) become connected points in the graph.

Each possible encounter requires some specific objects. There are four types of objects: *NPCs*, *containers*, *items*, and *NPC-pairs*. Some of the objects have labels so that two or more quest points can use the same object. In the *retrieve/deliver an item* pattern representation, the encounters for the *AcquireItem* and *DeliverItem* quest points both have item objects with the label *ItemFound*. This is to ensure that the PC delivers the same item that was acquired. Unless *SQUEGE* uses a label to denote that the same object should be used, no object is reused (sampling without replacement).

A possible encounter for the *ExposeQuest* point is to complete a sub-quest. When *SQUEGE* chooses this encounter it recursively generates this necessary sub-quest. The PC may only begin the original quest after completing this sub-quest. It is possible for the sub-quest to generate its own sub-quest as well; the result would be a sequence of quests. The game author can write a common theme for this sequence, making the quests feel like integral components of one long quest rather than many independent short quests.

SQUEGE prevents illogical quests from occurring by applying constraints at the level of encounters. Each encounter has preconditions and postconditions. The encounter for killing an NPC has a precondition that the NPC is alive and a postcondition that the NPC is dead. Selecting an encounter for a quest point involves checking the preconditions against a list of proven conditions that are given by previous quest points. If two conditions contradict each other then selecting that encounter fails and a new encounter is chosen. Two matching conditions leads to removing those conditions from the list of proven conditions. *SQUEGE* then adds the postconditions to the list of proven conditions – the new list is used as the proven conditions for the points that the quest point enables. This prevents the illogical quest described in Section 4.5, where the player tries to report the dragon's death to the dragon.

Currently *SQUEGE* supports the same five quest patterns found in ScriptEase: *retrieve/deliver an item*, *retrieve/deliver multiple items*, *retrieve/deliver one of multiple items*, *kill an NPC*, and *talk to*. Once a quest is added to the ScriptEase pattern catalogue, adding the same quest to *SQUEGE* is a straightforward process. Both the catalogue and *SQUEGE* have similar representations. The only major difference is that *SQUEGE* requires multiple encounters to be supplied for each quest point to produce a variety of quest instances. Therefore multiple ScriptEase quests with the same structure but different encounters could be combined into a *SQUEGE* pattern.

SQUEGE only gives an outline of the generated side-quest. An instance of the side-quest is easily created in ScriptEase. Before generating scripts, an author can adapt this instantiation by adding or removing situations and adding, removing, or changing the actions, definitions and conditions of the situations. Sometimes it is also useful to add or remove complete encounters. This flexibility allows instances to adapt well beyond their original pattern, giving the user a large amount of control.

Chapter 7

User Study

The previous chapter showed how *SQUEGE* can quickly and efficiently aid in the creation of side-quests for NWN, to create a more entertaining and immersive experience for the player. *SQUEGE* is only useful if it helps in producing high quality side-quests. This chapter describes a user study that aims to evaluate the quality of side-quests produced using *SQUEGE*.

7.1 What to Compare

To analyze the quality of the side-quests *SQUEGE* produces, its stories must be compared to stories that are known to be of high quality. The Aurora toolset is available to people who purchase NWN. This led to the creation of a large community of amateur game authors. Several websites exist that allow authors to distribute their NWN adventures to the public.

The Neverwinter Nights Vault website contains more than $5,064$ stories. Some of the highest rated NWN game adventures are found in the Neverwinter Nights Vault Hall of Fame [18]. One of these adventures is titled *Shadowlords* [15]. *Shadowlords* has been downloaded from the Neverwinter Nights vault $27,275$ times and has an average score of 8.92 out of 10.0. The initial section of *Shadowlords* contains a town in which the PC may perform four side-quests.

The *Shadowlords* adventure and its side-quests are considered by the community to be high in quality. Therefore, the quality of a set of side-quests can be evaluated by comparing the side-quests with those found in *Shadowlords*.

7.2 Preparation

To compare *SQUEGE* side-quests with the side-quests in *Shadowlords*, a mini-game adventure with side-quests similar to *Shadowlords* was prepared. This adventure was given the name *Darktide*.

Darktide was prepared by creating new side-quests for the initial section of *Shadowlords* with the aid of *SQUEGE*. This section of *Shadowlords* has a large number of NPCs, but the containers and items are limited to those only used in the side-quests and main story. Before generating the side-quests for *Darktide*, five containers and 13 items were added to *Darktide* so that random selections could be made during story generation. Therefore, *SQUEGE* had 41 NPCs, 24 containers, 17 items and 14 NPC pairs to use in its generated side-quests.

Since *Shadowlords* contains four side-quests, *SQUEGE* was used to generate six sets of side-quests, where each set contained four side-quests. To prevent biasing in favour of the *SQUEGE* side-quests, one of the six sets of side-quests was chosen at random. This set of side-quests was used in *Darktide*.

The side-quests in *Darktide* required a human author to create the dialogue for all of the conversations. The quality of the dialogue can be a major factor in determining the quality of the side-quests. To prevent this, the dialogue for *Darktide* was created by using phrases of dialogue from *Shadowlords* and replacing the nouns and verbs in the phrases with appropriate game objects and actions. The result is dialogue that is similar in structure to that found in *Shadowlords*.

Both adventures have the main story line removed from them. All that the PC can experience is the side-quests found in the initial town setting.

7.3 Side-Quest Descriptions

Both the test *Shadowlords* and *Darktide* stories have four side-quests each. This section gives a synopsis of each side-quest.

7.3.1 Shadowlords

This section describes the four side-quests used in *Shadowlords*.

The Missing Boy

Sarah asks the PC to search for Fynch, her son, who is lost. The PC finds Fynch in a local pawn shop owned by Angah Lalla. Fynch is trying to sell some goods that he stole. The PC finishes the side-quest by reporting Fynch's illegal activities to Sarah.

Love Potion No. 9

Preszmyr the herbalist's assistant accidentally sold a love potion to a woman named Anna who works at Ehaevaera's Beauty Rooms. Preszmyr has promised the PC a reward if the potion is retrieved. Anna has the potion, but refuses to return it. She is in love with Presephor, who works at the Thirsty Thunder Beast. The PC confirms that Presephor has feelings for Anna as well. After telling Anna of Presephor's feelings, Anna gives the PC the love potion. Finally the PC returns the potion to Preszmyr, who gives a generous reward.

A Taste of Intrigue

Gorn at the Thirsty Thunder Beast is raising Fire Beetles for Fistoon, the noodle vendor. Gorn asks the PC to clear the Fire Beetles out, as they have been causing him trouble. After killing the Fire

Beetles, the PC collects a reward from Gorn. However, there appears to be more to the story. The PC decides to investigate why Fistoon asked Gorn to keep the Fire Beetles. A member of the local militia, named Private Vilakon, suggests that proof of what Fistoon is using the Fire Beetles for may be found in Fistoon's apartment. A search of Fistoon's apartment yields a recipe for the noodles that Fistoon sells. The noodle recipe calls for Fire Beetle pancreatic juices. The PC reports this information to Private Vilakon, who can now administer justice.

A Missing Locket

A girl named Reela has lost her locket and asks the PC to find it. The PC is able to purchase the locket from Angah Lalla, a local pawnbroker. Finally the PC returns the locket to Reela, who is very grateful.

7.3.2 Darktide

The four side-quests used in *Darktide* are described in this section. Although the same NPC names and items are used in both *Shadowlords* and *Darktide*, they have no relationship from one story to the other. Note, during generation the second, third and fourth side-quests in *Darktide* were linked together. This linking resulted in the stories for all three side-quests being related.

Extra Help

The PC overhears that Fissten, a shopkeep, needs some extra help at the Steel Scabbard. After talking to Fissten, the PC is asked to find some Swiftwater boots and a Locket. Both items should be delivered to Ehaevaera at Ehaevaera's Beauty Rooms. The Swiftwater boots are found in the back room of Angah Lalla's. The locket was stolen from Ehaevaera's niece Reela. A local thief named Presephor should know where the locket is. After talking to Presephor, the PC discovers that the locket has been stashed in a nearby barrel. With both items found, the PC delivers them to Ehaevaera, who rewards the PC well.

Uncovering Secrets

Emmel, an agent of lord Kelemvor, asks the PC to discover the name of a person who is poisoning the local citizens. Finding the name involves the PC bribing Fistoon to reveal the poisoner's name. Fistoon directs the PC to a book which contains the names of several assassins. After acquiring the book, the PC brings the book back to Emmel, and she finally uncovers the poisoner's identity.

Confronting a Suspect

The name of the poisoner is Preszmyr the Herbalist. Since he is a member of the city council, Emmel cannot act against him. The PC is asked to confront Preszmyr instead. The PC confronts Preszmyr. Unfortunately, Preszmyr is not willing to turn himself in. Instead, Preszmyr attacks the PC. Before Preszmyr dies, he tells the PC that a patron at the Thirsty Thunder Beast has been sold a poisoned potion. The PC hurries to the Thirsty Thunder Beast to stop the patron from drinking the potion before it is too late. Unfortunately the patron drinks the potion before the PC arrives.

Creating a Cure

A patron of the Thirsty Thunder Beast has been poisoned and will die a slow and painful death. The PC asks Emmel to help create a cure. While Emmel gathers the necessary ingredients for a cure, the PC must find a flask to create the cure in. Pirchek, who is located near a house on the West side of the city, might know where to find a flask. Talking to Pirchek reveals that a local glassware merchant named Kessler might have some flasks. The PC purchases a Flask from Kessler and heads back to Emmel. The side-quest is finished when the PC gives the flask to Emmel. She is able to create the cure for the patron and the PC is rewarded for helping Emmel throughout the series of side-quests.

7.4 User Study Description

Question	Range of answers
This side-quest was humourous, funny, amusing	Strongly agree, disagree, neutral, agree, and strongly agree.
This side-quest was plausible, believable, self-consistent, coherent	Strongly agree, disagree, neutral, agree, and strongly agree.
This side-quest was surprising, suspenseful, eventful	Strongly agree, disagree, neutral, agree, and strongly agree.
This side-quest was complex, intricate	Strongly agree, disagree, neutral, agree, and strongly agree.
This side-quest was creative, novel	Strongly agree, disagree, neutral, agree, and strongly agree.
This side-quest was immersive, engaging, appropriately detailed, well paced	Strongly agree, disagree, neutral, agree, and strongly agree.
Overall I rate this side-quest as:	1, 2, 3, 4, 5; where 1 is poor and 5 is outstanding.

Table 7.1: The questions for the evaluation of a specific side-quest in the user study.

Between *Shadowlords* and *Darktide* there are eight side-quests. Four of the side-quests were created by a human author (Adam Miller), while four were created with the help of *SQUEGE*. To expedite the study, a class of university undergraduate students, most of whom had never played NWN, observed a demonstrator play through each of the stories. The demonstrator read the dialogue aloud to the students, used the same PC and showed no bias when demonstrating the game adventures. First the *Darktide* side-quests were demonstrated, then the *Shadowlords* side-quests were demonstrated. After the demonstrator completed each side-quest, the users were asked to complete a questionnaire. These questionnaires consisted of seven questions as well as space for comments. Table 7.1 lists each question and the range of possible answers for the side-quest questionnaires.

Question	Range of answers
Overall this story segment was consistent, coherent, connected, flowing	Strongly agree, disagree, neutral, agree, and strongly agree.
Overall this story segment was boring, repetitious, monotonous	Strongly agree, disagree, neutral, agree, and strongly agree.
Overall this story segment had conflict, obstacles, challenges, suspense	Strongly agree, disagree, neutral, agree, and strongly agree.
Overall this story segment had resolution or a purpose (point) or growth of protagonist	Strongly agree, disagree, neutral, agree, and strongly agree.
Overall I rate this story as:	1, 2, 3, 4, 5; where 1 is poor and 5 is outstanding.

Table 7.2: The questions for the evaluation of a specific story segment in the user study.

After all of the quests for *Darktide* had been demonstrated, an additional questionnaire about the overall adventure was completed by the users. The procedure was repeated with *Shadowlords*. The questionnaires ask the users to list the side-quests within the story segment from the best side-quest to the weakest side-quest. These questionnaires also contain five questions, as well as space for comments. Table 7.2 lists each question and the range of possible answers for the story segment questionnaires.

Once both game adventures had been demonstrated and the users had completed both the side-quest and story segment questionnaires, the users then completed a final questionnaire. This questionnaire asked the users to list their four favourite side-quests from either of the story segments, in order from first best to fourth best. The questionnaire also asked which story segment was preferred, *Darktide* or *Shadowlords*. Finally there was space on the questionnaire for the users to provide additional comments.

Due to time constraints of the 25 minute class period, only three of the four side-quests were shown with each game adventure. The three quests and the order they were demonstrated in *Darktide* were *Extra Help*, *Uncovering Secrets* and *Confronting a Suspect*. The three quests and the order they were demonstrated in *Shadowlords* were *The Missing Boy*, *Love Potion No. 9* and *A Taste of Intrigue*. The user study took place in a second-year education class. Before the user study, the students were told that the user study was voluntary, confidential and would have no impact on their grades. In total 72 students participated in the user study; 16 males, 42 females and 14 who did not specify a gender.

7.5 Results

Figure 7.1 shows how users ranked the side-quests within each story segment. In this Figure, the *Darktide* side-quests (shown in dark blue) are ranked independently from the *Shadowlords* side-quests (shown in light red). The theme of using blue for *Darktide* and red for *Shadowlords* will remain consistent for the rest of this chapter. Figure 7.1 gives a weighted ranking where the value

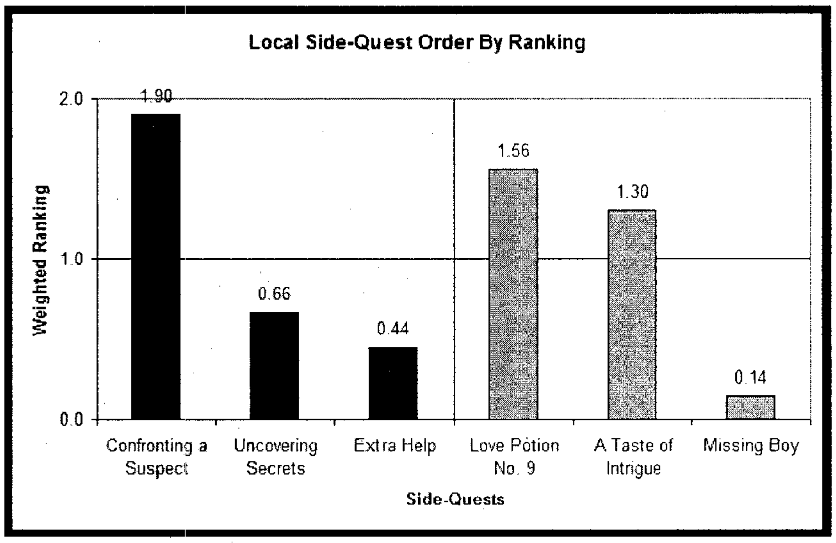


Figure 7.1: The weighted ranking of side-quests within each story segment.

given to a side-quest is determined using the formula:

$R_l = 2.0P_1 + 1.0P_2 + 0.0P_3$

where, R_l is the weighted rank value of the side-quest within its story segment, P_1 is the ratio of users that ranked the side-quest as the first best, P_2 is the ratio of users that ranked the side-quest as the second best and P_3 is the ratio of users that ranked the side-quest as the worst. For example, the maximum R_l of 2.0 means that 100% of the users ranked the quest as being the best quest of all three in the story segment.

Figure 7.1 shows that some side-quests were ranked higher than others. The *Darktide* side-quest *Confronting a Suspect* has a weighted ranking of 1.90, indicating that almost all of the users chose this side-quest as the best in *Darktide*. The rankings for the *Shadowlords* side-quests are different from the *Darktide* side-quests. Instead of having one side-quest stand out as ranking considerably higher than the others, a *Shadowlords* side-quest (*Missing Boy*) stands out as ranking considerably lower than the others (weighted ranking of 0.14). Almost all of the users ranked this side-quest as the worst of the side-quests in *Shadowlords*. The other two side-quests, *Love Potion No. 9* and *A Taste of Intrigue*, have weighted rankings significantly higher than *Missing Boy*. The reader should not interpret the 1.90 weighted ranking of the *Darktide* side-quest *Confronting a Suspect* as indicating that it ranks higher than any of the side-quest from *Shadowlords*. The 1.90 weighted ranking can only be compared to the other *Darktide* side-quests.

The next part of the user study asked the users to rank the four side-quests that they enjoyed the most. Figure 7.2 shows the weighted rank given to the side-quests. The formula for this global ranking is similar to the R_l shown before:

$R_g = 4.0P_1 + 3.0P_2 + 2.0P_3 + 1.0P_4 + 0.0P_5$

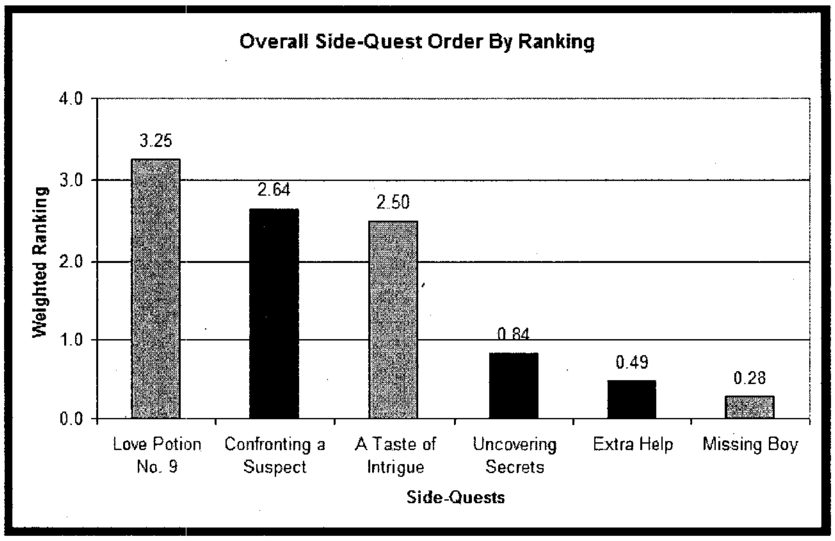


Figure 7.2: The weighted ranking of all of the side-quests.

where, R_g is the weighted rank value of the side-quest between all six side-quests, P_1 is the ratio of users that ranked the side-quest as the first best, P_2 is the ratio of users that ranked the side-quest as the second best, P_3 is the ratio of users that ranked the side-quest as the third best, P_4 is the ratio of users that ranked the side-quest as the fourth best and P_5 is the ratio of users that did not rank the side-quest as one of the four best side-quests.

This question only asks what the four most preferred side-quests are. A side-quest that consistently ranks as one of the three most preferred side-quests will have a large R_g, between 4.0 and 2.0. Conversely, a side-quest that consistently ranks as one of the three least preferred side-quests will have a low R_g, between 1.0 and 0.0, since the the formula gives little weight to side-quests ranked fourth and no weight to side-quests not ranked.

Figure 7.2 shows a separation between the rankings of the side-quests. Three of the side-quests consistently rank as being preferred: *Love Potion No. 9*, *Confronting a Suspect* and *A Taste of Intrigue*. The weighted rankings of these side-quests are between 3.25 and 2.50. The *Shadowlords* quest *Love Potion No. 9* is the most preferred side-quest. The next two preferred side-quests have similar weighted rankings; the *Darktide* side-quest *Confronting a Suspect* has a weighted ranking of 2.64 while the *Shadowlords* side-quest has a weighted ranking of 2.50.

The bottom three side-quests have weighted rankings between 0.84 and 0.28. The *Darktide* side-quest *Uncovering Secrets* received the fourth best weighted ranking of 0.84. This side-quest has a value close to 1.0 indicating that on average it was often ranked as the fourth most preferred side-quest. The *Darktide* side-quest *Extra Help* received a weighted ranking of 0.49. Finally the least preferred was the *Shadowlords* side-quest *Missing Boy* with a weighted ranking of 0.28.

These results indicate a type of layering between the quality of side-quests. All of the *Shad-*

owlords side-quests are not preferred over all of the *Darktide* side-quests, and vice versa. *Shadowlords* contains side-quests that are both the most and least preferred among all of the side-quests in the experiment.

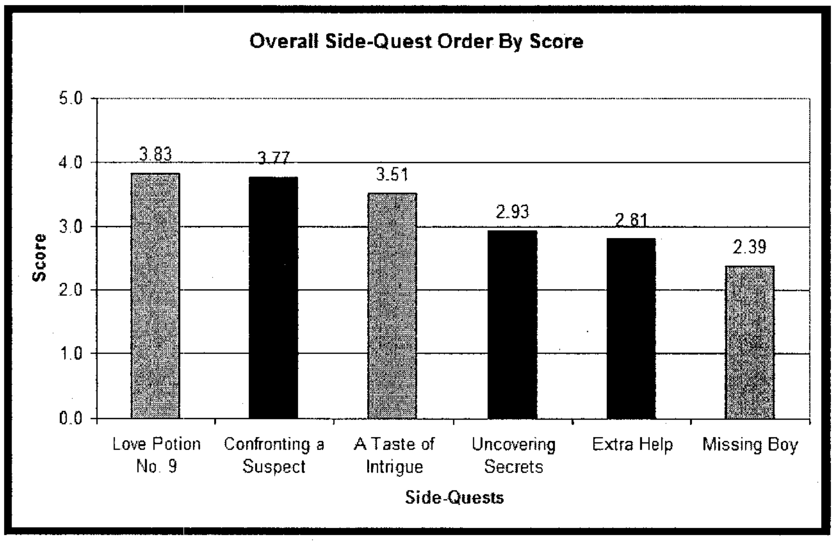


Figure 7.3: The average scores of each of the side-quests.

The same layering occurs when observing the average score of each side-quest. The users gave a score, between 1.0 and 5.0, to each quest. Figure 7.3 shows the average score of each side-quest. When examining the weighted rankings (Figure 7.2), there is a large difference between the two most preferred side-quests (*Love Potion No. 9* and *Confronting a Suspect*). The difference between the average scores of these two side-quests (Figure 7.3) is much smaller. *Love Potion No. 9* received an average score of 3.83 while *Confronting a Suspect* received an average score of 3.77.

A Taste of Intrigue received the third highest average score of 3.51. The fourth highest average score of 2.93 went to *Uncovering Secrets.* These two side-quests have a relatively large difference in scores. This reinforces the previous statement that the top three side-quests were consistently preferred over the bottom three side-quests.

There is also a relatively large difference between the average scores of the bottom two side-quests. *Extra Help* received an average score of 2.81 while *Missing Boy* received an average score of 2.39. This indicates that *Missing Boy* is the lowest quality side-quest, perhaps by a large margin.

The difference in quality between the two story segments appears to be that *Shadowlords* has two higher-quality side-quests and one lower-quality side-quest, while *Darktide* has one higher-quality side-quest and two lower-quality side-quests. However, the lower quality *Darktide* side-quests seem considerably higher in quality than the lower-quality *Shadowlords* side-quest (*The Missing Boy*).

Figure 7.4 shows the number of users that preferred each story segment. 10 users preferred *Darktide* while 55 users preferred *Shadowlords* and 12 users did not supply an answer. This indicates

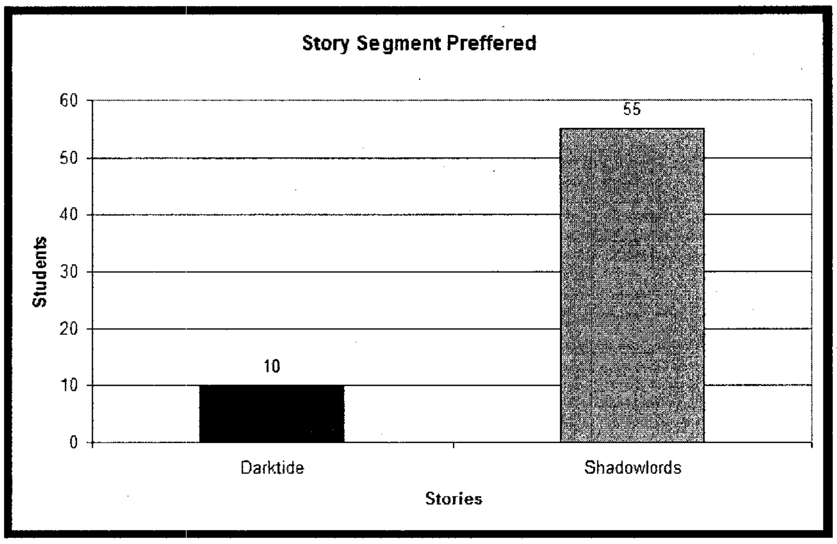


Figure 7.4: The number of users that preferred one story segment over another.

that the human authored story segment (*Shadowlords*) is higher in quality than the *SQUEGE* story segment (*Darktide*).

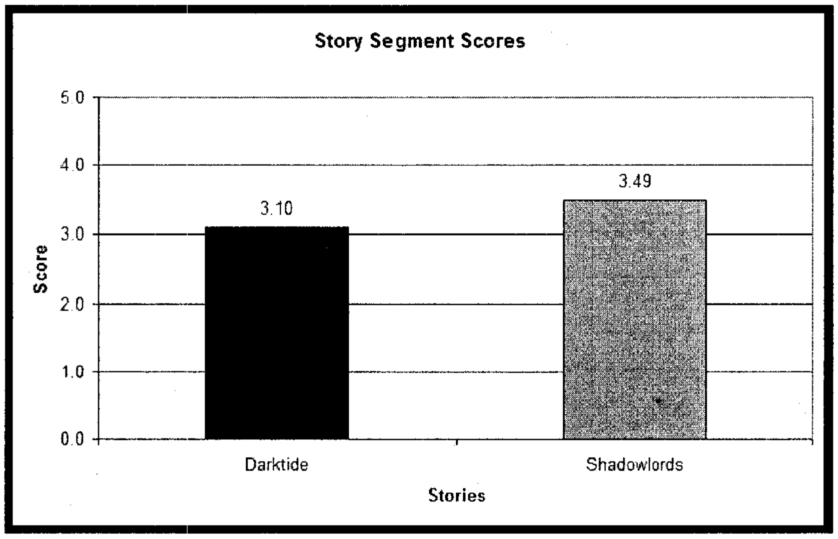


Figure 7.5: The average score of each story segment.

Figure 7.5 shows the average score that each story segment received. On a scale from 1 to 5, *Darktide* received an average score of 3.10 while *Shadowlords* received an average score of 3.49. The small difference between the two scores indicates that the difference in quality between the two story segments is relatively small. Most users preferred *Shadowlords* over *Darktide*. A t-test indicates that the difference between the two averages is significant ($t = 2.98, df = 123, p <$

0.002).

7.6 Limitations

The user study indicates that the *SQUEGE* side-quests were lower in quality than the human authored side-quests. The lower quality may be due to factors that are outside the domain of *SQUEGE*. This includes tasks that are the author's responsibility rather than *SQUEGE*'s.

The questionnaires invited the students to write comments about the side-quests. A common comment for the *Darktide* side-quests was that the dialogue was too linear. Normally dialogue in CRPGs is written so that there are many different choices as to what the PC may speak in the conversation. For example, in the *Shadowlords* side-quest *Love Potion No. 9*, the PC must have a conversation with Anna who has the love potion that the PC must acquire. Anna will not give the PC the love potion unless the PC can discover Presephor's feelings towards Anna, a man she has fallen in love with. At this point in the dialogue the PC has two choices to respond with. Agree to talk to Presephor or threaten Anna to give over the love potion. Choosing the latter choice results in Anna not believing the PC's threat and the PC must still talk to Presephor. Even though the different dialogue options did not change the outcome of the side-quest, they still add to the experience by giving the PC a feeling of an open-world.

It was known in advance that the side-quests would be demonstrated to the subjects rather than played by the subjects. Because of this, the conversations in the *Darktide* side-quests were designed with very few dialogue options. Many users noticed the lack of dialogue choices, as evident by their comments. The scores of the *Darktide* side-quests probably suffered because of this. If this user study is repeated in the future, the conversations in *Darktide* will be updated so that the average number of dialogue options for the PC is similar to the conversations in *Shadowlords*.

Another reason for the *Darktide* quests being lower in quality may be due to a lack of humour. The original *Shadowlords* adventures have some humour in the quests. For example, in *Love Potion No. 9*, if the PC is male then the PC has problems entering a beauty parlor because the parlor is restricted to women only. To enter the parlor the PC must fool the door guard into believing that the PC is a rather masculine woman. With *Darktide*, it was difficult to add humour to the dialogue while still having the phrases in the conversations remain similar in structure to the phrases in *Shadowlords*. This resulted in four side-quests with little humour.

One of the qualities that the users specified about each side-quest was how much humour the side-quest contained (Table 7.1). Figure 7.6 shows how humour correlates with the overall score of each of the side-quests. The correlation between the humour scores and the overall side-quest scores in *Darktide* is considerably smaller than the correlation between the humour scores and the overall side-quest scores in *Shadowlords*. This shows that humour was not a factor for the scores in the *Darktide* side-quests. Figure 7.7 gives the average score of perceived humour for each side-quest. This shows that humour helped improve the perceived quality of two of the side-quests in

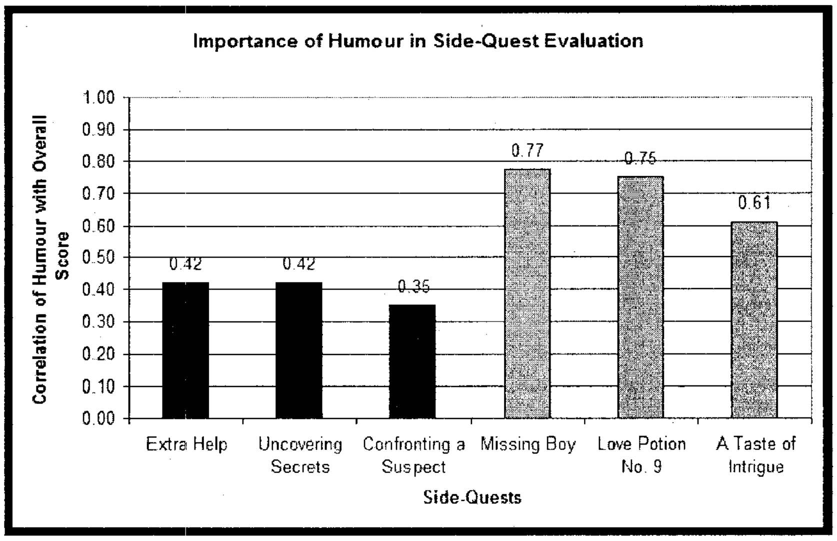


Figure 7.6: The correlation between humour and the overall score of each side-quest.

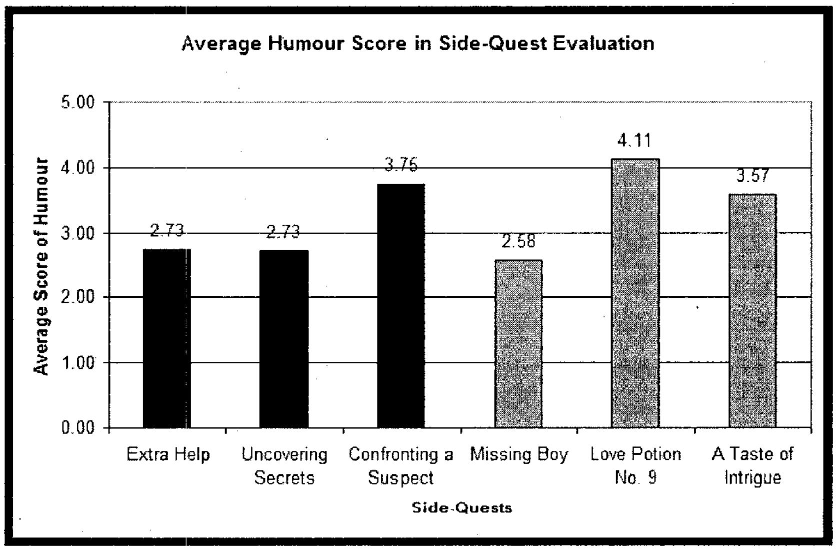


Figure 7.7: The average score of humour for each side-quest.

Shadowlords (*Love Potion No. 9* and *A Taste of Intrigue*). If more humour had been present in the *Darktide* side-quests then they may have received a better evaluation. Humour is a factor that is normally outside the domain of *SQUEGE*. Although humour can occur by accident. It is ultimately the author's decision to add humour to the side-quests.

Because of time constraints, the last side-quest in both *Darktide* and *Shadowlords* was not demonstrated. For *Darktide*, the *Creating a Cure* side-quest was not demonstrated. This is the final side-quest in a sequence of three related side-quests. *SQUEGE* generated the side-quests to be dependent upon each other. The first two side-quests *Uncovering Secrets* and *Confronting a Suspect* see the PC discovering the identity of Preszmyr who had been poisoning local citizens and then confronting Preszmyr. *Creating a Cure* concludes the side-quests by having the PC help in creating a cure for a victim who has been recently poisoned by Preszmyr. Because this final side-quest was not demonstrated, the story that spans all three side-quests was not concluded. This could have reduced the evaluations for the two preceding side-quests *Uncovering Secrets* and *Confronting a Suspect*, as some users may have felt there was no proper conclusion.

For *Shadowlords*, the side-quest *A Missing Locket* was not demonstrated. This side-quest is independent from all of the other side-quests in *Shadowlords*. Not demonstrating this quest did not lower the evaluations of any of the other side-quests in *Shadowlords*. If the same user study were to be repeated in the future, different side-quests would be excluded. *Extra Help* would be excluded from *Darktide* and *The Missing Boy* would be excluded from *Shadowlords*. Both side-quests are independent from the other side-quests in their respective stories, and both were the lowest ranked and scored side-quests in their stories.

When using *SQUEGE*, if the author does not approve of the side-quests generated, the side-quests can be adapted to the author's specifications or new side-quests can be generated in their place. For *Darktide* this was not the case. The design of the experiment was to use the side-quest outlines given without alterations. Presumably, the *Shadowlords* side-quests were created such that the author did approve of the side-quests created, otherwise the side-quests would not be in Shadowlords. Any game author using *SQUEGE* would be able to alter and remove side-quests at will, improving the quality of the final side-quests produced.

Despite all of these limitations, *Darktide* received a reasonably favourable evaluation. With a good author, *SQUEGE* should help produce high-quality side-quests both efficiently and quickly for use in a commercial CRPG.

Chapter 8

Future Work and Conclusions

This chapter presents a summary of this work, discusses some future work and concludes with some closing remarks.

8.1 Summary

The stories found in CRPGs can be quite complex due to the interactive nature of CRPGs. The PC in a CRPG can simultaneously follow multiple story lines. These story lines (or quests) can also have multiple resolutions. Not only must game authors have a way of organizing the quests in CRPGs, but they must also be able to change and query the game state to have the quests function properly. The current technology for controlling and querying game state is to use a scripting language. It is difficult to track and maintain the scripts and global variables associated with a quest, as most CRPGs locate scripts according to the event that the script is associated with. Quests involve the collaboration of scripts located throughout various events in the game adventure. Scripting also requires the game author to have knowledge of computer programming or to employ someone with that knowledge. It is not uncommon for commercial CRPGs to have hundreds or thousands of quests. Some of these quests are short, simple adventures, called side-quests, that are independent from the main story line. While they are independent from the main story, side-quests are important as they add a sense of freedom to the game, increasing the player's entertainment. Creating the large number of side-quests needed for a commercial CRPG can be a tedious task that consumes a large amount of resources. Often game authors will take a short-cut and exclude or reduce the number of side-quests in a game or simplify them, resulting in less game content.

This work focused on two applications. First, the concept of quest patterns was introduced. Quest patterns, which are similar to design patterns in software engineering [8], provide a way to specify quests. The various encounters that the PC may have during the quest are structured through using quest points. Each quest point specifies what action or actions the PC performs, the preconditions for the point, and what happens in response to the PC's actions. The result is a complete and straightforward specification of a quest. While each quest in a CRPG is unique, many share common similarities. By modeling these similarities using patterns, a game author can create specifications from these patterns with a small amount of adaptation. This work shows that quest specifications are detailed enough that an automated scripting tool, such as ScriptEase [13], is able to use quest patterns to generate the required scripts for those quests in the commercial CRPG Neverwinter Nights (NWN) [16]. A case study was performed that replaced the scripts required for all the quests in a set of areas from the official campaign for NWN with quest patterns. The case study showed that quest patterns require less effort to use than writing the scripts for the quests by hand.

The second application of this work deals with aiding game authors in the rapid creation of side-quests for CRPGs. Because side-quests are short and simple, they can be instantiations of a quest pattern with a minimal number of adaptations. This work presented the Sub-QUEst GEnerator (*SQUEGE*), a tool that generated outlines of sub-quests for CRPGs. *SQUEGE* uses a modified version of quest patterns. Each point in a *SQUEGE* quest pattern has a weighted list of encounters that *SQUEGE* can select from. As input, *SQUEGE* takes a list of objects from the game adventure to be used in the side-quests, and a list of *SQUEGE* quest patterns. *SQUEGE* then generates a specified number of side-quest instances by using the patterns. For each point in a pattern, one of the point's multiple encounters is selected using the weights of each encounter to bias the selection. Objects for each encounter are assigned by randomly selecting an object from the input. *SQUEGE* also checks for logical inconsistencies, to prevent the generated side-quests from functioning improperly. As output, *SQUEGE* gives the game author a graphical outline of the side-quests. The outline is specific enough that the game author can quickly create the necessary conversations and scripts. Alternatively, ScriptEase can also be used by the author to generate the scripts. If the game author is not satisfied with the outlines created by *SQUEGE*, the process can be repeated and an entirely different set of side-quests will be generated. To evaluate the effectiveness of *SQUEGE*, a user study was performed. The study had two sets of side-quests being demonstrated to a class of university undergraduate students. Both sets of side-quests used the same setting, but one set was created with *SQUEGE* while the other was extracted from a popular NWN game adventure created by a human author. The students were asked to evaluate each set of side-quests. The results indicated that the *SQUEGE* side-quests were slightly lower in quality to the human-authored side-quests. Several limitations outside the domain of *SQUEGE* (e.g., adding humour and complexity to conversations) were identified. The quality of the side-quests created with the aid of *SQUEGE* should increase with a good author.

8.2 Future Work

The current quest pattern catalogue in Appendix A is rather small. There are only five patterns in the catalogue and three are variants of the *retrieve/deliver an item* pattern. The intent of the current catalogue is to show that quest patterns work. The catalogue needs to be expanded to include more coverage for other types of quests commonly found in CRPGs. One quest that could be added to the catalogue is where the PC must find a way to open a locked door. The quest could have multiple resolutions: finding the key to the door, picking the lock on the door, bribing a nearby guard to open the door or bashing the door open. Another type of quest pattern could see the PC finishing a sequence of quests. The pattern would be responsible for determining when new quests are available for the PC to begin. For example, the PC may have two quests to complete: one quest requires the character to act lawfully by bringing a criminal to justice, and the other quest requires the PC to perform unlawful actions by stealing from a king's treasury. Finishing the lawful quest first has the PC starting a line of quests working as a member of the king's personal guard. Conversely, finishing the unlawful quest starts a sequence of quests where the PC joins a group of bandits. Determining when these sequences of quests begin can be done inside each quest. However, allowing the author to manage the quests in a higher-level quest pattern allows the details of the quest sequences to be independent from the quests themselves.

Currently *SQUEGE* only generates an outline of side-quests. The game author still has to create the conversations and scripts. For NWN, the conversations are created in the Aurora Toolset while the scripts can be easily created with ScriptEase. In the future, *SQUEGE* will be extended to not only provide an outline but also instantiating each side-quest in ScriptEase. Each quest point in an outline created by *SQUEGE* specifies an encounter with associated objects. Each point in the outline also specifies what its *enablers* are. From this information the side-quests can be automatically instantiated in ScriptEase. Placeholder conversations will also be added to the appropriate NPCs. Points in the conversations will be labeled to indicate which quest points they enable. The game author can then adapt the conversation and side-quest instances as needed.

Currently *SQUEGE* generates side-quests that are static; once the game adventure begins, the side-quests cannot change. The PC may then perform an action that prevents certain side-quests from being completed. For example, a *SQUEGE* generated side-quest sees the player starting the side-quest by talking to an NPC named Dwight. However, the PC decides to kill Dwight before talking to him. Since the side-quest is static, this action invalidates it as the PC can no longer begin the side-quest. *SQUEGE* could be changed so that the generated side-quests are dynamic. If a side-quest is invalidated, after the game has begun, the side-quest can dynamically change so the player can still complete it. With the example where Dwight is killed, the side-quest can reassign Dwight's role to another NPC in the game. *SQUEGE* could be made dynamic by using techniques similar to the Hierarchical Task Networks (HTN) [4] discussed in Section 2.3. In an HTN, when a task fails the system is able to backtrack to another task. *SQUEGE* could generate selected quest points with multiple encounters. Should the first encounter become impossible to complete, the next encounter is what the PC must perform in order for the quest point to become reached.

8.3 Conclusions

This work provides two important contributions. First, it supplies a method for organizing and maintaining plot in CRPGs. Most CRPGs do not have tools to aid in the design of quests. With quest patterns, it is easier to design the story lines so that game authors can create more interesting and complex stories. Second, is the creation of *SQUEGE* to help alleviate the time and resources needed to create the optional side-quests found in CRPGs. This allows game authors and programmers to focus their efforts on other tasks such as designing the main story line.

Bibliography

[1] Bioware Corp. http://www.bioware.com/, 2007.

[2] J. Campbell. *The Hero with a Thousand Faces.* Princeton University Press, 1973.

[3] M. Carbonaro, M. Cutumisu, M. McNaughton, C. Onuczko, T. Roy, J. Schaeffer, D. Szafron, S. Gillis, and S. Kratchmer. Interactive Story Writing in the Classroom: Using Computer Games. In *Proceedings of the International DiGRA Conference 2005*, pages 323–338, Vancouver, BC, 2005.

[4] M. Cavazza, F. Charles, and S. Mead. Character-Based Interactive Storytelling. *IEEE Intelligent Systems*, 17(4):17–24, 2002.

[5] M. Cutumisu, D. Szafron, J. Schaeffer, M. McNaughton, T. Roy, C. Onuczko, and M. Carbonaro. Generating Ambient Behaviors in Computer Role-Playing Games. *IEEE Intelligent Systems*, 21(5):88–99, 2006.

[6] C. Fairclough and P. Cunningham. A Multiplayer Case Based Story Engine. In *GAME-ON*, pages 41–46, London, UK, 2003.

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

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series. Addison-Wesley, 1995.

[9] G. Gygax and D. Arneson. *Dungeons and Dragons.* Tactical Studios Rules and Wizards of the Coast, 1974-2003.

[10] Jade Empire. http://jade.bioware.com/.

[11] Jade Empire (Limited Edition), Game Credits. http://www.mobygames.com/game/c64/ultima-iv-quest-of-the-avatar/credits%.

[12] 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.

[13] M. McNaughton, M. Cutumisu, D. Szafron, J. Schaeffer, J. Redford, and D. Parker. ScriptEase: Generative Design Patterns for Computer Role-Playing Games. In *Proceedings of the 19th IEEE Conference on Automated Software Engineering (ASE 2004)*, pages 88–99, Linz, Austria, September 2004.

[14] J. Meehan. TALE-SPIN – An Interactive Program that Writes Stories. In *Proceedings of the 5th International Joint Conference on AI*, pages 91–98, Cambridge, MA, 1977.

[15] A. Miller. Shadowlords, Dreamcatcher, and Demon Campaigns. http://adamandjamie.com/nwn/, 2007.

[16] Neverwinter Nights. http://nwn.bioware.com/.

[17] Neverwinter Nights: An Introduction to the Plot Wizard. http://nwn.bioware.com/builders/plotwizard_intro.html.

[18] Neverwinter Nights Vault - Hall of Fame. http://nwvault.ign.com/View.php?view=Modules.HOF, 2007.

[19] E. Packard. *Choose Your Own Adventure series*. Bantam Books, New York, 1979-89.

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

[21] M. Riedl, H. Lane, R. Hill, and W. Swartout. Automated Story Direction and Intelligent Tutoring: Towards a Unifying Architecture. In *Proceedings of the 2005 AIED Workshop on Narrative Learning Environments*, Amsterdam, 2005.

[22] M. Riedl and R. Young. From Linear Story Generation to Branching Story Graphs. *IEEE Journal of Computer Graphics and Applications*, pages 23–31, May-June 2006.

[23] Star Wars. http://www.imdb.com/title/tt0076759/, 1977.

[24] Star Wars: Episode V - The Empire Strikes Back. http://www.imdb.com/title/tt0080684/, 1980.

[25] Star Wars: Episode VI - Return of the Jedi. http://www.imdb.com/title/tt0086190/, 1983.

[26] Ultima IV: Quest of the Avatar, Game Credits. http://www.mobygames.com/game/c64/ultima-iv-quest-of-the-avatar/credits%.

[27] Unreal Technology Overview. http://www.unrealtechnology.com/html/technology/ue30.shtml, 2007.

Appendix A

Quest Pattern Catalogue Specifications

This appendix gives specifications for each of the quest patterns created. Each quest point given provides several preset options, such as a journal entry and experience awarded. The values given are generalized for the pattern. When a pattern is used these options should be changed to the specific needs of the user. Some quest points have no encounters associated with them. In these cases, the quest point becomes reached as soon as the point becomes available the number of times specified.

A.1 Retrieve/deliver an item quest

- Intent: *The player is asked to first acquire an item and then retrieve or deliver the item.*
- Options:
  - *Quest Giver Conversation*
  - *Item*
  - *Item Receiver*
- Quest Points
  - *ExposeQuest*
    - Intent: *A placeholder to expose the quest. Currently does nothing but can be expanded by adding an encounter such as overhearing a conversation.*
    - Options:
      - Quest Point Type: *Normal*
      - Enablers: ⟨*Initially Available*⟩
      - Journal Entry: ⟨*None*⟩
      - Experience Awarded: *0*
    - Encounters:
      - Having the quest point become available at least *1* time(s)
    - When quest point reached:
      - *No additional actions required*
  - *TalkToQuestGiver*

    * Intent: *The player reaches a specific point in a conversation where they are asked to acquire an item and then retrieve or deliver it.*
    * Options:
        - Quest Point Type: *Normal*
        - Enablers: *ExposeQuest*
        - Journal Entry: *You have been asked to retrieve/deliver an item.*
        - Experience Awarded: *0*
    * Encounters:
        - Reaching the *Quest Giver Conversation*
    * When quest point reached:
        - *No additional actions required*
- *FindItemLocation*
    * Intent: *A placeholder to discover the item location. Currently does nothing but can be expanded by adding an encounter such as reaching a specific point in a conversation where the player is told the item location.*
    * Options:
        - Quest Point Type: *Normal*
        - Enablers: *TalkToQuestGiver*
        - Journal Entry: ⟨*None*⟩
        - Experience Awarded: *0*
    * Encounters:
        - Having the quest point become available at least *1* time(s)
    * When quest point reached:
        - *No additional actions required*
- *AcquireItem*
    * Intent: *The player acquires the item to be retrieved/delivered.*
    * Options:
        - Quest Point Type: *Normal*
        - Enablers: *TalkToQuestGiver, FindItemLocation*
        - Journal Entry: *You have acquired the item that is to be retrieved/delivered.*
        - Experience Awarded: *0*
    * Encounters:
        - The player acquires *Item*
    * When quest point reached:
        - *No additional actions required*
- *GiveItem*
    * Intent: *The player retrieves/delivers the item to its receiver.*
    * Options:
        - Quest Point Type: *Close*
        - Enablers: *AcquireItem*
        - Journal Entry: *The item has been retrieved/delivered.*
        - Experience Awarded: *0*
    * Encounters:
        - *Item Receiver* acquires *Item*
    * When quest point reached:
        - *No additional actions required*

A.2 Retrieve/deliver multiple items quest

- Intent: *The player is asked to first acquire multiple items and then retrieve or deliver the items.*
- Options:
  - *Quest Giver Conversation*
  - *Item 1*
  - *Item 2*
  - *Items Receiver*
- Quest Points
  - *ExposeQuest*
    - Intent: *A placeholder to expose the quest. Currently does nothing but can be expanded by adding an encounter such as overhearing a conversation.*
    - Options:
      - Quest Point Type: *Normal*
      - Enablers: ⟨*Initially Available*⟩
      - Journal Entry: ⟨*None*⟩
      - Experience Awarded: *0*
    - Encounters:
      - Having the quest point become available at least *1* time(s)
    - When quest point reached:
      - *No additional actions required*
  - *TalkToQuestGiver*
    - Intent: *The player reaches a specific point in a conversation where they are asked to acquire some items and then retrieve or deliver them.*
    - Options:
      - Quest Point Type: *Normal*
      - Enablers: *ExposeQuest*
      - Journal Entry: *You have been asked to retrieve/deliver some items.*
      - Experience Awarded: *0*
    - Encounters:
      - Reaching the *Quest Giver Conversation*
    - When quest point reached:
      - *No additional actions required*
  - *FindItemLocation1*
    - Intent: *A placeholder to discover the first item location. Currently does nothing but can be expanded by adding an encounter such as reaching a specific point in a conversation where the player is told the item location.*
    - Options:
      - Quest Point Type: *Optional*
      - Enablers: *TalkToQuestGiver*
      - Journal Entry: ⟨*None*⟩
      - Experience Awarded: *0*
    - Encounters:
      - Having the quest point become available at least *1* time(s)
    - When quest point reached:

    - *No additional actions required*
- *AcquireItem1*
  - Intent: *The player acquires the first item to be retrieved/delivered.*
  - Options:
    - Quest Point Type: *Optional*
    - Enablers: *TalkToQuestGiver, FindItemLocation1*
    - Journal Entry: *You have acquired an item.*
    - Experience Awarded: *0*
  - Encounters:
    - The player acquires *Item 1*
  - When quest point reached:
    - *No additional actions required*
- *FindItemLocation2*
  - Intent: *A placeholder to discover the second item location. Currently does nothing but can be expanded by adding an encounter such as reaching a specific point in a conversation where the player is told the item location.*
  - Options:
    - Quest Point Type: *Optional*
    - Enablers: *TalkToQuestGiver*
    - Journal Entry: ⟨*None*⟩
    - Experience Awarded: *0*
  - Encounters:
    - Having the quest point become available at least *1* time(s)
  - When quest point reached:
    - *No additional actions required*
- *AcquireItem2*
  - Intent: *The player acquires the second item to be retrieved/delivered.*
  - Options:
    - Quest Point Type: *Optional*
    - Enablers: *TalkToQuestGiver, FindItemLocation2*
    - Journal Entry: *You have acquired an item.*
    - Experience Awarded: *0*
  - Encounters:
    - The player acquires *Item 2*
  - When quest point reached:
    - *No additional actions required*
- *AcquireMultipleItems*
  - Intent: *The player acquires all the items to be retrieved/delivered.*
  - Options:
    - Quest Point Type: *Normal*
    - Enablers: *AcquireItem1, AcquireItem2*
    - Journal Entry: *You have acquired all the items.*
    - Experience Awarded: *0*
  - Encounters:
    - Having the quest point become available at least 2 time(s)
  - When quest point reached:

            - *No additional actions required*
  - *GiveItems*
    - Intent: *The player retrieves/delivers the items to their receiver.*
    - Options:
      - Quest Point Type: *Close*
      - Enablers: *AcquireMultipleItems*
      - Journal Entry: *The items have been retrieved/delivered.*
      - Experience Awarded: *0*
    - Encounters:
      - *Items Receiver* acquires both *Item 1* and *Item 2*
    - When quest point reached:
      - *No additional actions required*

A.3 Retrieve/deliver one of multiple items quest

- Intent: *The player is asked to acquire one of several items and then retrieve or deliver that item.*
- Options:
  - *Quest Giver Conversation*
  - *Item 1*
  - *Item 2*
  - *Item 1 Receiver*
  - *Item 2 Receiver*
- Quest Points
  - *ExposeQuest*
    - Intent: *A placeholder to expose the quest. Currently does nothing but can be expanded by adding an encounter such as overhearing a conversation.*
    - Options:
      - Quest Point Type: *Normal*
      - Enablers: ⟨*Initially Available*⟩
      - Journal Entry: ⟨*None*⟩
      - Experience Awarded: *0*
    - Encounters:
      - Having the quest point become available at least *1* time(s)
    - When quest point reached:
      - *No additional actions required*
  - *TalkToQuestGiver*
    - Intent: *The player reaches a specific point in a conversation where they are asked to acquire one of multiple items and then retrieve or deliver that item.*
    - Options:
      - Quest Point Type: *Normal*
      - Enablers: *ExposeQuest*
      - Journal Entry: *You have been asked to retrieve/deliver one of multiple items.*
      - Experience Awarded: *0*

    * Encounters:
        - Reaching the *Quest Giver Conversation*
    * When quest point reached:
        - *No additional actions required*
- *FindItemLocation1*
    * Intent: *A placeholder to discover the first item location. Currently does nothing but can be expanded by adding an encounter such as reaching a specific point in a conversation where the player is told the item location.*
    * Options:
        - Quest Point Type: *Optional*
        - Enablers: *TalkToQuestGiver*
        - Journal Entry: ⟨*None*⟩
        - Experience Awarded: *0*
    * Encounters:
        - Having the quest point become available at least *1* time(s)
    * When quest point reached:
        - *No additional actions required*
- *AcquireItem1*
    * Intent: *The player acquires the first item to be retrieved/delivered.*
    * Options:
        - Quest Point Type: *Optional*
        - Enablers: *TalkToQuestGiver, FindItemLocation1*
        - Journal Entry: *You have acquired an item.*
        - Experience Awarded: *0*
    * Encounters:
        - The player acquires *Item 1*
    * When quest point reached:
        - *No additional actions required*
- *FindItemLocation2*
    * Intent: *A placeholder to discover the second item location. Currently does nothing but can be expanded by adding an encounter such as reaching a specific point in a conversation where the player is told the item location.*
    * Options:
        - Quest Point Type: *Optional*
        - Enablers: *TalkToQuestGiver*
        - Journal Entry: ⟨*None*⟩
        - Experience Awarded: *0*
    * Encounters:
        - Having the quest point become available at least *1* time(s)
    * When quest point reached:
        - *No additional actions required*
- *AcquireItem2*
    * Intent: *The player acquires the second item to be retrieved/delivered.*
    * Options:
        - Quest Point Type: *Optional*
        - Enablers: *TalkToQuestGiver, FindItemLocation2*

        - Journal Entry: *You have acquired an item.*
        - Experience Awarded: *0*
    - Encounters:
        - The player acquries *Item 2*
    - When quest point reached:
        - *No additional actions required*
  - *GiveItem1*
    - Intent: *The player retrieves/delivers the first item to its receiver.*
    - Options:
        - Quest Point Type: *Close*
        - Enablers: *AcquireItem1*
        - Journal Entry: *You have retrieve/deliverd an item.*
        - Experience Awarded: *0*
    - Encounters:
        - *Item 1 Receiver* acquires *Item 1*
    - When quest point reached:
        - *No additional actions required*
  - *GiveItem2*
    - Intent: *The player retrieves/delivers the second items to its receiver.*
    - Options:
        - Quest Point Type: *Close*
        - Enablers: *AcquireItem2*
        - Journal Entry: *You have retrieved/delivered an item.*
        - Experience Awarded: *0*
    - Encounters:
        - *Item 2 Receiver* acquires *Item 2*
    - When quest point reached:
        - *No additional actions required*

A.4 Talk to quest

- Intent: *The player is asked to talk to another character.*
- Options:
  - *Quest Giver Conversation*
  - *NPC Target Conversation*
  - *NPC Target*
- Quest Points
  - *ExposeQuest*
    - Intent: *A placeholder to expose the quest. Currently does nothing but can be expanded by adding an encounter such as overhearing a conversation.*
    - Options:
        - Quest Point Type: *Normal*
        - Enablers: ⟨*Initially Available*⟩
        - Journal Entry: ⟨*None*⟩

        - Experience Awarded: *0*
    - Encounters:
        - Having the quest point become available at least *1* time(s)
    - When quest point reached:
        - *No additional actions required*
- *TalkToQuestGiver*
    - Intent: *The player reaches a specific point in a conversation where they are asked to talk to another NPC.*
    - Options:
        - Quest Point Type: *Normal*
        - Enablers: *ExposeQuest*
        - Journal Entry: *You have been asked to talk to an NPC.*
        - Experience Awarded: *0*
    - Encounters:
        - Reaching the *Quest Giver Conversation*
    - When quest point reached:
        - *No additional actions required*
- *FindNPCLocation*
    - Intent: *A placeholder to discover the NPC location. Currently does nothing but can be expanded by adding an encounter such as reaching a specific point in a conversation where the player is told the NPC location.*
    - Options:
        - Quest Point Type: *Optional*
        - Enablers: *TalkToQuestGiver*
        - Journal Entry: ⟨*None*⟩
        - Experience Awarded: *0*
    - Encounters:
        - Having the quest point become available at least *1* time(s)
    - When quest point reached:
        - *No additional actions required*
- *TalkToNPCTarget*
    - Intent: *The player talks to the NPC target.*
    - Options:
        - Quest Point Type: *Close*
        - Enablers: *TalkToQuestGiver, FindNPCLocation*
        - Journal Entry: *You have talked to the NPC.*
        - Experience Awarded: *0*
    - Encounters:
        - Reaching the *NPC Target Conversation*
    - When quest point reached:
        - *No additional actions required*
- *NPCTargetDies*
    - Intent: *The death of the NPC Target completes the quest.*
    - Options:
        - Quest Point Type: *Close*
        - Enablers: *TalkToQuestGiver*

            · Journal Entry: *The NPC has died, you can no longer talk to them.*
            · Experience Awarded: *0*
        * Encounters:
            · *NPC Target* dies
        * When quest point reached:
            · *No additional actions required*

A.5 Kill a creature quest

- Intent: *The player is asked to kill a creature.*
- Options:
    - *Quest Giver Conversation*
    - *Creature Killed*
    - *Reporter Conversation*
- Quest Points
    - *ExposeQuest*
        * Intent: *A placeholder to expose the quest. Currently does nothing but can be expanded by adding an encounter such as overhearing a conversation.*
        * Options:
            · Quest Point Type: *Normal*
            · Enablers: ⟨*Initially Available*⟩
            · Journal Entry: ⟨*None*⟩
            · Experience Awarded: *0*
        * Encounters:
            · Having the quest point become available at least *1* time(s)
        * When quest point reached:
            · *No additional actions required*
    - *TalkToQuestGiver*
        * Intent: *The player reaches a specific point in a conversation where they are asked to kill a creature.*
        * Options:
            · Quest Point Type: *Normal*
            · Enablers: *ExposeQuest*
            · Journal Entry: *You have been asked to kill a creature.*
            · Experience Awarded: *0*
        * Encounters:
            · Reaching the *Quest Giver Conversation*
        * When quest point reached:
            · *No additional actions required*
    - *FindCreatureLocation*
        * Intent: *A placeholder to discover the creature location. Currently does nothing but can be expanded by adding an encounter such as reaching a specific point in a conversation where the player is told the item location.*
        * Options:
            · Quest Point Type: *Normal*

        - Enablers: *TalkToQuestGiver*
        - Journal Entry: ⟨*None*⟩
        - Experience Awarded: *0*
    - Encounters:
        - ⟨*None*⟩
        - Having the quest point become available at least *1* time(s)
    - When quest point reached:
        - *No additional actions required*
- *KillCreature*
    - Intent: *The player kills the creature.*
    - Options:
        - Quest Point Type: *Normal*
        - Enablers: *TalkToQuestGiver, FindCreatureLocation*
        - Journal Entry: *You have killed the creature.*
        - Experience Awarded: *0*
    - Encounters:
        - *Creature Killed* dies
    - When quest point reached:
        - *No additional actions required*
- *ReportCreatureDead*
    - Intent: *The player reports that the creature is dead.*
    - Options:
        - Quest Point Type: *Close*
        - Enablers: *KillCreature*
        - Journal Entry: *You have reported that the creature is dead.*
        - Experience Awarded: *0*
    - Encounters:
        - Reaching the *Reporter Conversation*
    - When quest point reached:
        - *No additional actions required*