Automatic Story Generation for Computer Role-Playing Games

Curtis Onuczko, Duane Szafron, Jonathan Schaeffer, Maria Cutumisu, Jeff Siegel,
Kevin Waugh and **Allan Schumacher**

Department of Computing Science, University of Alberta
Edmonton, Canada
{onuczko, duane, jonathan, meric, siegel, waugh, schumach}@cs.ualberta.ca

Abstract

Scripting the plot in a computer role-playing game requires a large number of scripts that are difficult to track and maintain. Game adventures often have simple plots, called sub-quests, that are independent from the main plot. Sub-quests are important, as they add value to the open-world appeal of the game, but they still have to be scripted. We have created a prototype of a tool that helps by automatically producing design pattern specifications for sub-quests. The specifications can be entered into an existing tool, called ScriptEase, to generate scripting code for Neverwinter Nights adventures, without doing any manual scripting. The sub-quest patterns produced are logically consistent, ensuring the story can be completed by the player. The sub-quests are also designed to produce a better story by having the author adjust the amount of interactivity between the sub-quests. The entire process is done with little input from the author.

Scripting Sub-Quests With Patterns

The plot in computer role-playing games (CRPGs) has become quite complex. It is common for CRPGs to have complex story lines that allow for multiple story-arcs. Plot in CRPGs also involves the player character (PC) interacting with multiple game objects over time. Examples of interaction include the PC having a conversation with a character, unlocking a door, or acquiring an item. All of these interactions must be scripted, requiring expensive programmer time.

Often in CRPGs, a simple plot will occur that is independent of the main story line. These simple plots, called sub-quests, are usually included to give the player a sense of a vast open world with unlimited opportunities and are often optional. Sub-quests are of vital importance to CRPGs. In the most popular games, a player can choose to take a break from the main plot and perform an optional sub-quest at any time.

Sub-quests greatly add to the breadth of choices in a CRPG and provide motivations for a player to explore the game world. This increases a player's sense of customization and ownership of PCs. The influence of the player on the main plot is often constrained, as specific plot options are necessary to keep the game on track. Without any sub-quests, the player will feel constrained in their ability to influence the story. With sub-quests, the player can make more unconstrained choices, since the main plot is protected.

Sub-quests are generally simple compared to the complexity of the main plot, but CRPGs will often have a large number of sub-quests. Several sub-quests that all have a small amount of complexity can exhibit connections among them and maintain recurring patterns. These recurring patterns provide a theme that increases motivation, excitement and a sense of importance to activities that would otherwise be meaningless in the game.

An example sub-quest pattern is to *retrieve an item*. This involves the PC needing to *retrieve an item* for some purpose. Examples of the *retrieve an item* pattern include the PC being asked by a non-player character (NPC) to retrieve a stolen amulet, the PC encountering a locked door requiring that a key be retrieved to unlock it, and the PC needing to retrieve a gear in order for a machine to work.

A pattern can be represented as a sequence of roles. Each role in the pattern is played by an object in the game other than the PC. A role also has an action associated with it. This action represents the interaction between the PC and the object associated with the role that must take place in order for the role to be fulfilled. Examples of roles are: reaching a specific point in a conversation with an NPC, unlocking a door, and acquiring an item. Figure 1 shows a sequence of roles for a sample *retrieve an item* pattern.

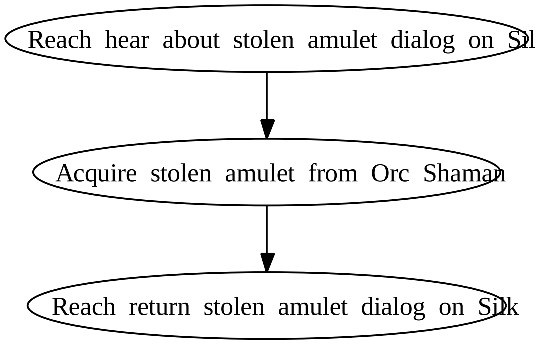
Reach hear about stolen amulet dialog on Sil
Acquire stolen amulet from Orc Shaman
Reach return stolen amulet dialog on Silk

Figure 1: An example *retrieve an item* pattern.

These roles can also serve as a specifications for programmers. Each role in the plot pattern corresponds to a script that has to be written. Each script handles the action between the PC and the object in each role. Each script also contains control structures that ensure the PC must fulfill each role in the same order as they are presented in the pattern.

Using a tool such as ScriptEase (McNaughton *et al.* 2004), allows plot patterns to be represented by generative design patterns. A user would then only have to instantiate pattern instances and all of the scripts for the sub-quest would be generated automatically. Other systems generate scripts at a low level, while ScriptEase uses patterns to generate scripts at a high level of abstraction. This allows complex stories to be created quickly and efficiently.

Generating Sub-Quests

A plot pattern for a sub-quest only requires an object and an action to be specified for every role. Once these objects and actions are specified, the pattern can be easily transformed into scripts. The objects and actions can be automatically chosen such that the process of producing a sub-quest pattern instance requires no creative input from the user.

Our sub-quest generator (*SQUEGE*) is written in prolog and generates sub-quest pattern instance specifications. A user needs to provide a list of objects available in the game that can be used in the sub-quests. *SQUEGE* then logically chooses the objects and actions needed to specify a sub-quest pattern, by ensuring that the resulting sub-quest can be completed by the player. This process can then be repeated multiple times to generate the number of patterns desired.

It is important for the generated sub-quests to be logical, otherwise the player will become quite frustrated at attempting to follow sub-quests that are impossible to complete. An example of an illogical sub-quest would be a *retrieve an item* quest, where the player needs to acquire a key to unlock a chest, and the key is inside the chest. Generating sub-quests requires each pattern to have constraints that must be satisfied. The constraint that would solve the illogical example mentioned previously would be to ensure that the key being acquired is not located in the chest that needs to be unlocked.

Generated sub-quests can also interact with each other to create a more engaging experience for the player. Two sub-quests interact with each other if they share one or more identical roles. An example of this would be a role that occurs at the end of a sub-quest and at the beginning of another sub-quest. This would require the player to complete the first sub-quest before being allowed to start on the second sub-quest. This interaction between sub-quests makes sub-quests more complex. The increase in complexity enriches the player's experience as the sub-quests available are not just a set of simple independent activities.

Rather than having interactivity between sub-quests being a by product of the sub-quests generated, *SQUEGE* can add more or less sub-quest interactivity based on heuristics. The interactivity can be adjusted depending on a number given to the program that specifies the amount of interactivity desired between sub-quests. A low number results in most quests being independent while a high number results in most quests interacting with one another.

Demonstration Overview

The demonstration creates a game adventure in BioWare Corp.'s popular CRPG, Neverwinter Nights. The game adventure takes place in a castle setting commonly found in fantasy games. The adventure lacks a main plot so as to focus on sub-quests, and will initially be unscripted.

A list of all the game objects that can be scripted are given to *SQUEGE*. The initial states of the game objects, such as whether a door is locked or the location of an item, are also given to *SQUEGE*. *SQUEGE* then outputs a specification of the sub-quest patterns generated. Figure 2 shows the output of a generated *retrieve an item* pattern instance.

```
Pattern: Retrieve an item
  Roles:
    Assigner: Reach hear about stolen
      amulet dialog on Silk
    Resolver: Acquire stolen amulet from
      Orc Shaman
    Closer: Reach return stolen amulet
      dialog on Silk
  Transitions:
    Transition from Assigner to Resolver
    Transition from Resolver to Closer
```

Figure 2: Output for a *retrieve an item* pattern.

When a pattern role involves a conversation, the intent of the conversation is given, but the conversation itself must be written by the author. *SQUEGE* will tell the author that the PC must reach a point in the conversation, but does not specify the location or the text, only the intent (context) of the conversation, such as retrieve the magic ring.

In the demo, one of the sub-quest pattern specifications will be instantiated using ScriptEase. Although *SQUEGE* only generates a sub-quest pattern specification, manually instantiating a sub-quest pattern in ScriptEase from a specification is simple and straightforward. The demo will then show how ScriptEase generates the scripts corresponding to the outline. The demo will end with the sub-quest being played in Neverwinter Nights. Additional game adventures that have all of their sub-quest pattern specifications instantiated before the demonstration will be available for viewing.

References

McNaughton, M.; Cutumisu, M.; Szafron, D.; Schaeffer, J.; Redford, J.; and Parker, D. 2004. Scriptease: Generative design patterns for computer role-playing games. In *ASE*, 88–99. IEEE Computer Society.