Motivational Ambient and Latent Behaviors in Computer RPGs

Maria Cutumisu, Duane Szafron, Jonathan Schaeffer, Kevin Waugh

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

Abstract

Character behaviors in computer role-playing games have a significant impact on game-play, but are often difficult for game authors to implement and adapt. We present a behavior model that requires no manual script writing. In this model, behaviors can be interrupted and resumed, they can transition to other behaviors depending on game events, and they can be chosen based on motivations. We have extended the generative pattern approach of ScriptEase to support behaviors. We describe an implementation of this model that generates scripting code for a commercial game, *Neverwinter Nights*.

Introduction

Behaviors for non-player characters (NPCs) in computer role-playing games (CRPGs) have been traditionally implemented using scripting languages. Although these languages are supposed to be "higher-level" than programming the game engine directly, they are similar in scope and abstraction level to C or C++. The increasing complexity of behaviors makes manual scripting impractical, especially for collaborative NPCs. Often game authors are not programmers; they rely on programmers for scripting which can delay development and introduce errors. Other issues hinder the proliferation of complex NPC behaviors in commercial games, such as consistency, scalability, and robustness (Spronck *et al.* 2006).

We use a guard NPC to illustrate our behavior model. The guard performs realistic actions such as patrolling, resting, checking that the guarded item remains in its chest, and conversing with other guards. At the end of the shift, the guard goes to a tavern, talks to other patrons and orders drinks. If the guarded item is stolen, the guard abandons the empty chest and flees the area to avoid facing the "boss" (Movies 2007). To create richer behaviors in thousands of background NPCs, we need to eliminate manual scripting. We must quickly and reliably create reusable behaviors adaptable to a wide range of NPCs.

We implement our behavior model using ScriptEase (McNaughton *et al.* 2003) patterns. ScriptEase has previously been used to represent a basic set of independent ambient behaviors and simple collaborative behaviors implemented using encounter patterns (Cutumisu *et al.* 2006). We introduce for the first time native support for behavior patterns in ScriptEase. This research extends the power of behavior patterns in four important ways. First, it adds latent behaviors, including the ability to return to partially-completed ambient behaviors after a latent behavior is completed. Second, it adds behavior roles, a mechanism that allows the NPC to change behavior models during the story. Third, it provides a motivation model for selecting ambient behaviors. Finally, it introduces a novel collaboration protocol that simplifies collaborations across a broader range of NPCs.

Behavior Model

The components of our model work together to provide behaviors that are expressive for the needs of CRPGs and intuitive for game authors to understand. We highlight each term in our ontology in *italics* the first time it appears and again when defined. We distinguish NPC behaviors on two axes, *independent* vs. *collaborative* and *ambient* vs. *latent*. An ambient behavior is *proactive* or *reactive*.

An *independent* behavior is performed alone, while a *collaborative* behavior is performed jointly with a collaborating NPC. A collaborative behavior must include concurrency control to ensure that the actions of the two NPCs are synchronized. An *ambient* behavior is spontaneously initiated by an NPC (*proactive*) or performed in response to a collaborative behavior initiated by another NPC (*reactive*). When an NPC finishes a proactive behavior, it selects another proactive behavior based on motivations. This selection process continues indefinitely or until the NPC is interrupted by an external event. To respond to a collaborator on a topic, an NPC must perform a *reactive* behavior on the same topic. A game event may interrupt the NPC while it is performing an ambient behavior. A *latent* behavior (independent or collaborative) is an NPC's reaction to an appropriate *cue* (external event). After the latent behavior is completed, the NPC should return to the interrupted ambient behavior. A reactive behavior may be reused, being triggered by either a proactive or a latent collaborative behavior. An NPC may exhibit different *roles* at different times, containing all of the behaviors (both ambient and latent) that can be performed in a particular context. The active role of an NPC is changed by a cue. For example, our guard NPC uses a **Travel** role during a specific time interval each day. When the guard finds the item missing, a cue changes its role to **Flee** and the guard leaves the area.

Behavior Patterns

We have captured the guard's behaviors and other common NPC behaviors into ScriptEase *behavior patterns*, a category of reusable generative design patterns (Gamma *et al.* 1994).

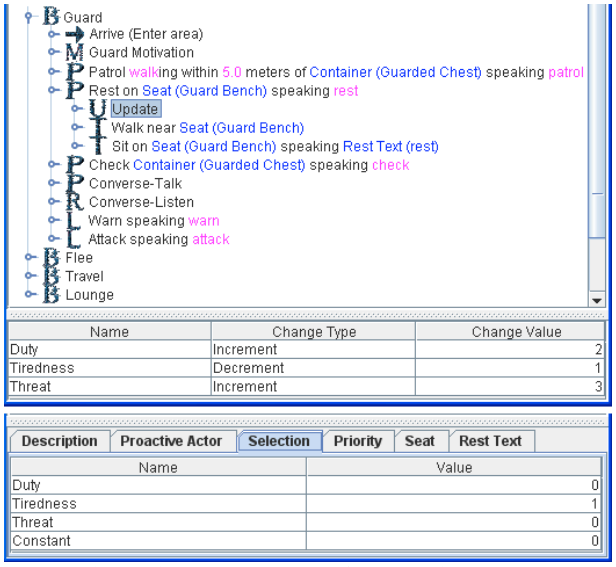

Figure 1. The structure of a behavior pattern.

To use an existing behavior pattern (e.g., **Guard**), an author only needs to instantiate the pattern and set its options. However, to extensively adapt existing patterns or to create new patterns using pre-built components, an author must understand the structure of behavior patterns. In Figure 1, the **Guard** instance has been opened to reveal some of its components. The **Guard** role (**B**) includes a cue (➞), **Arrive**, to activate this behavior when the guard enters the area of the container, a motivation (**M**), **Guard Motivation**, three independent ambient behaviors (**P**), **Patrol**, **Rest** and **Check**, one collaborative ambient behavior (**P**), **Converse-Talk**, one collaborative reactive ambient behavior (**R**), **Converse-Listen**, and two independent latent behaviors (**L**), **Warn** and **Attack**. Each proactive, reactive and latent behavior is based on a ScriptEase *abstract behavior*. When instantiating an abstract behavior, the designer selects the particular kind of cue that triggers it and that determines which kind of behavior it will be: proactive (**P**), reactive (**R**) or latent (**L**). Once it is instantiated, it is referred to as a *basic behavior*. A basic behavior consists of its cue (➞), an update clause (**U**) that updates the motivation, and a set of *tasks* (**T**) composed of *actions* (**A**). Tasks ensure atomicity and synchronization within a collaborative behavior. If a proactive behavior is interrupted by a latent behavior, the NPC restarts the interrupted task when the latent behavior is finished. A *cue* (➞) is used to activate a role (the **Guard** role is activated when the NPC enters the area of the container) or to trigger a basic behavior (the **Warn** latent behavior is triggered by a range-based **Near** cue). Each proactive behavior is triggered by a motivation-based cue, while a reactive behavior is triggered by a reactive-based cue initiated by the collaborator. Our cue-based timing mechanism, general purpose cues (based on game state) and motivational scheme for specifying NPC behaviors allow us to express all of the semantics provided by the character AI in *Oblivion*. Our generative pattern technique allows the game author to rely on a catalog of pre-defined behavior patterns that generate scripts automatically.

Every role contains a set of proactive behaviors. The NPC selects a proactive behavior based on motivation (**M**) attribute values that are stored in a motivation vector and updated as the game progresses. The **Guard Motivation** contains three attributes: **Duty**, **Tiredness**, and **Threat**. **Duty** refers to the NPC's sense of duty and it can change during the guard's shift. When an ambient behavior (**Rest**) is completed, its motivation vector is updated (**Tiredness** is reduced, while **Duty** and **Threat** are increased, as shown in the **Rest** update clause of Figure 1). The selection value for a proactive behavior is a constant plus the dot product of the *motivation vector* and the *proactive vector*. The current selection value for the **Rest** behavior, $S(\boldsymbol{Rest}) = c^{\boldsymbol{Rest}} + \boldsymbol{w}^{\text{Rest}} \bullet \boldsymbol{m}$, is shown at the bottom of Figure 1. The *motivation vector* contains the current values of the motivation attributes. The *proactive vector* contains one weight, w_i, for each motivation attribute that reflects how important this attribute is in selecting the motivation.

Collaborative Behaviors

In our model, an NPC resumes an interrupted ambient behavior after completing a latent behavior. Moreover, an NPC can perform an ambient behavior while waiting for its collaborator to complete a lengthy behavior. We have developed a simple collaborative system that authors can use to specify abstract behaviors that can constitute proactive and reactive components in a collaborative behavior. For example, a collaborative behavior between a guard and a friendly NPC can be established if the guard has a proactive behavior (**Converse-Talk**) on a topic ("weather") and the collaborator has a reactive behavior (**Converse-Listen**) on the same topic. The collaborative model looks for any NPC who has a reactive behavior on that topic, makes *eye-contact* (it is not busy in a collaborative or latent behavior) and satisfies the conditions of the initiator's collaborative behavior. Both NPCs concurrently execute pairs of tasks that comprise their respective behaviors until completion. Scripting code that synchronizes these tasks is generated automatically.

Behavior Dispatch

The key to our behavior model is dispatching the appropriate behavior at any time and remembering what behavior to return to when an interrupting behavior is complete.

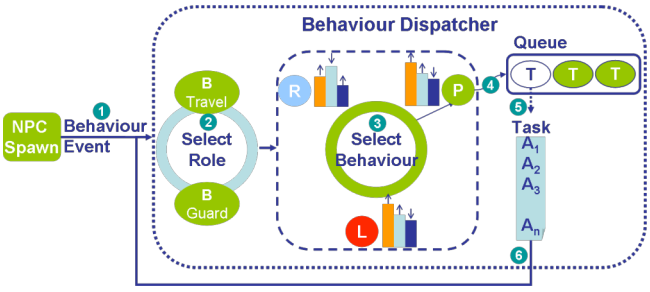

Figure 2. Behavior dispatch in the behavior model.

When an NPC is created in the game, a custom behavior event is triggered on the NPC, as illustrated in Figure 2. Then, a role is selected using a cue and the role invokes a spinner that may choose a proactive behavior based on the role's motivation. This new proactive behavior triggers a set of tasks composed of the actions that implement the behavior. At any time, an NPC can be performing a latent (independent or collaborative), a collaborative (proactive or reactive), or an independent behavior. An NPC has four queues: one for latent behaviors, two for ambient collaborative behaviors and one for ambient independent behaviors. Each queue holds the tasks not yet finished for that basic behavior. The spinner first tries to select and perform a task from one of the queues. If it is successful, it dequeues the task and calls the role selector. The spinner picks tasks from the queues in this priority order: latent, collaborative, independent. When picking a task from a collaborative queue, the spinner checks if the collaborator has signalled the start of the next task. If not, the spinner waits for a short time and checks again. If it still has not been signalled, it proceeds to the next ambient queue or selects a new proactive behavior. When a task from the collaborative queue is dequeued, the spinner signals the collaborator that it can proceed to its next task. If the collaborator has not finished, the waiting NPC cancels the collaboration, since the collaborator may be blocked. This ensures no deadlock or indefinite postponement for NPCs. Both NPCs spin again and may execute the same protocol.

Behavior Pattern Evaluation

We conducted experiments to evaluate the usability of behavior patterns for non-programmers and the scalability of our model. We asked 24 high school students with no programming experience to use ScriptEase patterns to write an interactive story as part of their tenth-grade English curriculum. The ScriptEase version contained a library of 27 behavior patterns and 60 encounter patterns. In the 4.5 hours they were allowed to author their interactive stories, 9 out of 24 students used 13 behavior instances of 6 behavior patterns. Students found behavior patterns easy to use, but were restricted by the time they were allotted to complete the many aspects of the story. To overcome this time restriction, we asked a high-school summer student to write an interactive story. She used 40 instances of 21 behavior patterns for 202 NPCs. The scene ran flawlessly with no degradation or perceptible delays.

To validate the expressiveness of our behavior patterns, we took less than an hour to replace all the manual scripts for the ambient characters of the *Prelude* module of the *NWN* official campaign with the code generated from our behavior patterns. This exercise showed that the new behavior approach is efficient, robust, and easy to use.

Conclusions

We designed our behavior system to satisfy the computational and functional requirements of commercial computer games (Spronck *et al.* 2006). Our model and implementation are *robust*, *flexible*, *extendable*, and *scalable* to thousands of ambient NPCs (hundreds active at a time), while requiring minimal CPU resources. Our NPC behaviors are *clear*, since the intent of ScriptEase patterns is evident from their names and from an inspection of their components. ScriptEase generates commented scripts for NPC behaviors that are easily interpretable even for non-programmers and allows the specification of behaviors at a higher level, making adaptation, experimentation, and testing easy. Our system provides *variety* in selecting roles and in choosing motivational proactive behaviors, and supports the generation of unpredictable, but rational and *consistent* behaviors. We developed a concurrency mechanism that solves the synchronization problems of collaborative, interruptible and resumable behaviors and we can generate engaging behaviors without writing code.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Institute for Robotics and Intelligent Systems (IRIS), and Alberta's Informatics Circle of Research Excellence (iCORE). We thank our University of Alberta Women in Scholarship, Engineering, Science and Technology (WISEST) student, Stephie Cadek, along with the other current ScriptEase team members, Curtis Onuczko, Jeff Siegel, Allan Schumacher, and Robin Miller.

References

Cutumisu, M.; Szafron, D.; Schaeffer, J.; McNaughton, M.; Roy, T.; Onuczko, C.; Carbonaro, M. 2006. Generating Ambient Behaviors in Computer Role-Playing Games. *IEEE Journal of Intelligent Systems* 21(5) 19-27.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

McNaughton, M.; Redford, J.; Schaeffer, J.; Szafron, D. 2003. Pattern-based AI Scripting using ScriptEase. *Canadian AI*, 35-49.

Movies. 2007. http://www.cs.ualberta.ca/~script/ movies/aiide07.

Spronck, P.; Ponsen, M.; Sprinkhuizen-Kuyper, I.; Postma, E. 2006. Adaptive Game AI with Dynamic Scripting. *Machine Learning* 63(3): 217-248.