A Demonstration of ScriptEase II

Matthew Church[1], Eric Graves[1], Jason Duncan[1], Adel Lari[1], Robin Miller[1], Neesha Desai[1], Richard Zhao[1], Mike Carbonaro[1], Jonathan Schaeffer[1], Nathan Sturtevant[2], Duane Szafron[1]

[1]Department of Computing Science, University of Alberta

[2]Department of Computer Science, University of Denver

{mfchurch, graves, jtduncan, lari, remiller, neesha, rxzhao, mcarbona, jonathan, dszafron}@ualberta.ca, sturtevant@cs.du.edu

Abstract

This demonstration describes ScriptEase II, a tool that allows game story authors to generate scripts that control objects in video games by manipulating high level story patterns and game objects. ScriptEase II can generate scripting code for any game engine for which a translator is written. Currently there are translators for Neverwinter Nights and real Pinball games.

Story Creation With ScriptEase II

ScriptEase (Cutumisu et al. 2007) allows authors to generate scripting code for the Neverwinter Nights (NWN) video game. However, designing a scripting tool on a per-game-engine basis is costly. ScriptEase II can generate code for any game engine, using interchangeable "translators". Two game engines at opposite ends of the genre spectrum were chosen, to verify the code generation system – NWN (Aurora game engine) and a modified network version of USC's pinball API (Wong et al. 2010). Authors use a drag and drop interface to assemble event driven scenarios using cause-effect patterns that can be abstracted across games and genres. For example, pinball is more similar to a game like NWN one might initially expect, since pinball games often involve quest elements by asking players to hit various targets on the board.

In story-based games, a creature reacts when something happens to it. Suppose an author wants to describe how a *Bodak* will react when it is hit by a spell: first gasp, and then, if the *Bodak* possesses a particular item (*Briarspike*), speak a second line and attack, otherwise speak a warning. Figure 1 illustrates how an author can generate scripts for this scenario using ScriptEase II. The author drags a *cause*, *When [subject] becomes the target of (Spell) cast by (Spell Caster)*, from the library pane to the story pane (see inset). The author changes the category in the library pane from *causes* to *effects* (not shown) and drags an *effect*, *Speak Text [ ] at volume [ ]*, from the library pane to the story pane. The author types *AHHHH* in the "brown" text slot and selects a speech volume, *Shout*, from the "brown" volume slot. A circle indicates the kind of object that can be placed in the slot – *C* for *creature*, *P* for *placeable*, *S* for *spell* and *T* for text. The author drags the "blue" *Bodak* (*creature*) from the game object pane (lower left) to the *[subject]* slot so that the *Bodak* will respond to being hit.

The author uses a question to specify the conditional effects, by dragging the *effect*, *? [ ],* from the *effect* category of the library pane to the story pane (see inset). To fill in the question slot with a specific question, the author changes the category in the library pane to *descriptions* (not shown) and drags a *description*, *HasItem describes if [creature] has [ ]*, to the story pane (see inset). The author then drags *Bodak* from the top line of the story pane to the *[subject]* slot and drags the *Briarspike* item object from the game object pane to the *[ ]* slot. The author then drags the *HasItem* description from the *description* line to the *question* line. The question has two parts, *yes* and *no* and the author drags different *effects* to each of the parts. During gameplay, the *yes* or the *no* effects are performed depending on whether the *Bodak* has the *Briarspike*. Figure 1 shows the complete story pattern.

The "green" objects, *Spell* and *Spell Caster* are dynamic game objects that the clause presents to the author. The author can drag a "green" object from the cause to any effect. For example, the author wants the *Bodak* to attack the creature who cast the spell. This object is not known during story creation. It becomes known during game-play when a creature actually casts a spell at the *Bodak*. Each *cause* has a custom set of objects that become available to the author when that cause activates.

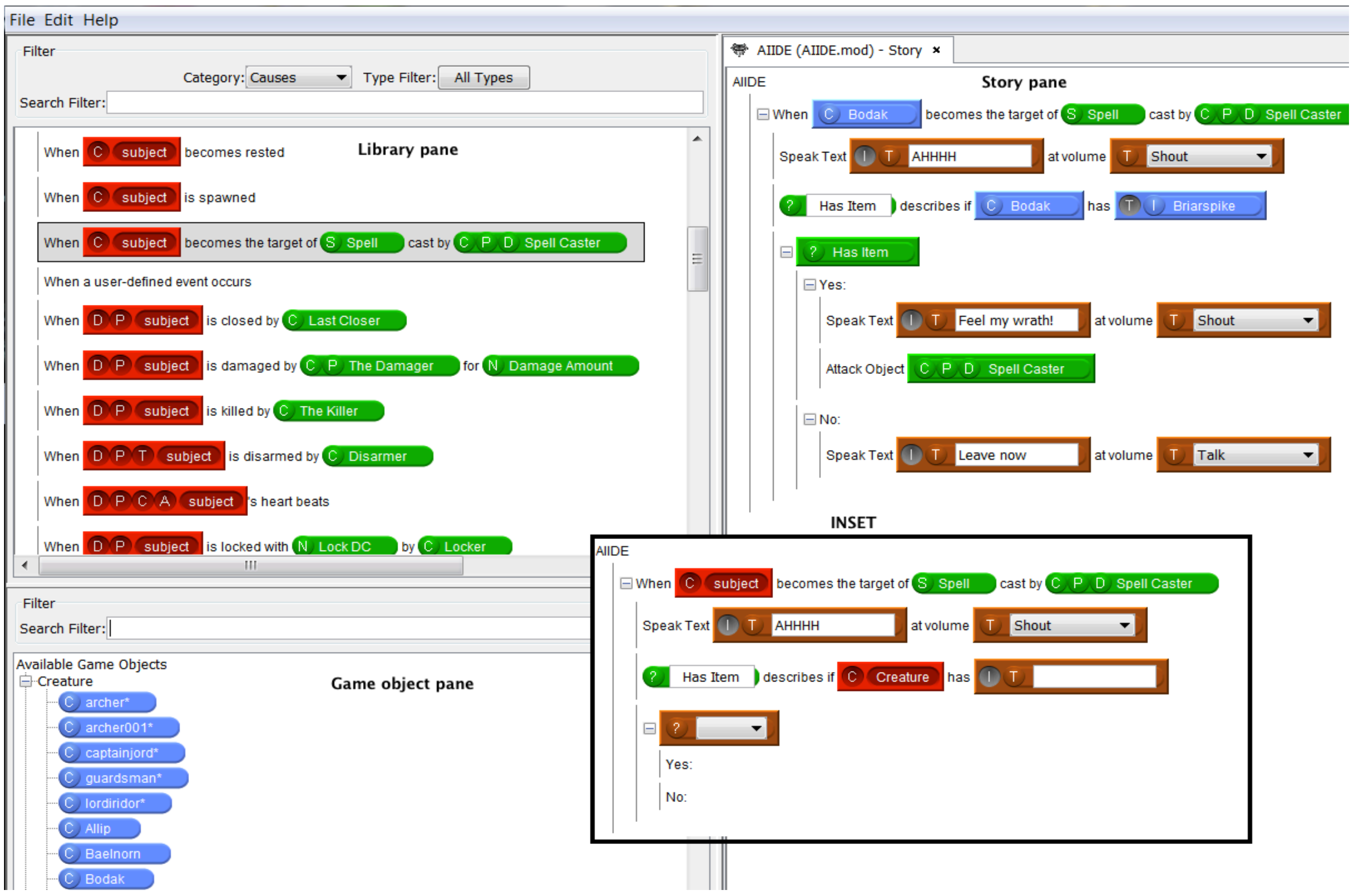
File Edit Help
Filter
Category: Causes
Type Filter: All Types
Search Filter:
Library pane
When C subject becomes rested
When C subject is spawned
When C subject becomes the target of S Spell cast by C P D Spell Caster
When a user-defined event occurs
When D P subject is closed by C Last Closer
When D P subject is damaged by C P The Damager for N Damage Amount
When D P subject is killed by C The Killer
When D P T subject is disarmed by C Disarmer
When D P C A subject 's heart beats
When D P subject is locked with N Lock DC by C Locker
Filter
Search Filter:
Available Game Objects
Creature
Game object pane
C archer*
C archer001*
C captainjord*
C guardsman*
C lordiridor*
C Allip
C Baelnorn
C Bodak
AIIDE (AIIDE.mod) - Story
AIIDE
Story pane
When C Bodak becomes the target of S Spell cast by C P D Spell Caster
Speak Text I T AHHHH at volume T Shout
? Has Item describes if C Bodak has T I Briarspike
? Has Item
Yes:
Speak Text I T Feel my wrath! at volume T Shout
Attack Object C P D Spell Caster
No:
Speak Text I T Leave now at volume T Talk
INSET
AIIDE
When C subject becomes the target of S Spell cast by C P D Spell Caster
Speak Text I T AHHHH at volume T Shout
? Has Item describes if C Creature has I T
?
Yes:
No:

Figure 1 A screenshot from ScriptEase II, with an inset that shows some open slots.

The author can save a story and ScriptEase II generates scripts for the target game engine and writes them to the appropriate files for that engine. In addition, ScriptEase II stores the story patterns in a ScriptEase II file so that they can be edited and used to regenerate scripting code

If an author wants to save a story component in a library, then that component is dragged from the story pane to the library pane and it is stored in a library file for use in other stories. When a story component is *promoted* to the library, all story specific objects are removed from the component. For example, the author could save the contents of the story pane in Figure 1 to the library and the Bodak, Briarspike and text fields would become empty slots. Note that the "green" *Spell Caster* and *HasItem* objects would not be removed from the *creature* and *question* slots respectively, since they do not depend on game objects available in a specific story. A demo movie is available at: www.cs.ualberta.ca/~script/AIIDE2011-SEII.mov.

Code Generation

ScriptEase II code generation uses interchangeable game "translators" to create scripting code for a variety of game engines ranging from standard video games to real game machines like pinball. These translators are a combination of XML files, which specify attributes such as the types of game objects, the available API functions, and the events activated during gameplay.

To write a translator, the format of the game files and the API of the game engine must be known. In addition to the XML libraries in the translator, an interface must be provided for reading and writing to the desired story modules. Story modules often define the unique game objects available for that particular story, and are in a format readable by the game engine. For example in NWN, the story module would contain the creatures, placeables, doors, etc. that appear in a particular story. Pinball is a special case for code generation since there exists a single story module, as every story contains the same available switches, lamps and solenoids.

References

Cutumisu, M., et al. ScriptEase: A Generative/Adaptive Programming Paradigm for Game Scripting. *Science of Computer Programming*, 67 (1), June 2007, 32-55.

Wong, D., et al. Implementing Games on Pinball Machines. In *Foundations of Digital Games (FDG)*, 2010, 240-247.