Descriptions - a viable choice for video game authors

Neesha Desai
Department of Computing Science
University of Alberta
neesha@ualberta.ca

Duane Szafron
Department of Computing Science
University of Alberta
dszafron@ualberta.ca

ABSTRACT

Modern video game development activities have become as specialized as movie-making activites. Gifted story-writers, artists, and animators have replaced programmers in most content creation activities. However, there is still one area where computer programmers play a big role. Stories, characters, and events are still controlled by scripts that are written in "C-like" languages. Therefore, scripting the video game content usually requires a high level of programming knowledge. Some scripting is simple, such as specifying specific game objects. However, in order to take advantage of knowledge learned during game play, authors need to be able to specify dynamic game objects. This often requires authors to create complex definitions, which are composed of a series of variable assignments in programming languages. In this paper, we show how these definitions can be replaced by a more natural mechanism, which we call descriptions. We also present the results of a user study that shows that authors with no programming skills can use descriptions more effectively than definitions and that the authors prefer descriptions.

Categories and Subject Descriptors

D.3.3 [**Software**]: Programming Languages—*General*

1. INTRODUCTION

Video games have become a major part of our culture. The games continue to become longer and more complicated. Many games now have multiple ways to complete the game, which can provide the player with a difference experience each time they play the game. Tools like Alice [2], Storytelling Alice [3], and Scratch [6] are helping to make it possible for non-programmers to create their own animated interactive stories and simple video games.

While more people are becoming interested in creating their own video games, the tools being used by professional game designers are not accessible for the average non-programmer. More and more video games are including authoring tools as part of the game package, like *Neverwinter Nights* [1] or as the game itself, like *Kodu* [7]. However, many of these games require the author to learn a scripting language specific to the game, like NWScript for *Neverwinter Nights* or TES Script used in the *The Elder Scrolls (TES)* construction kit [4]. These scripting languages are usually similar to C or Java. Tools like Scratch, Alice, and Kodu are easier to use, but are not capable of expressing the complex character behaviors seen in popular commercial games.

When designing a video game, it is easy for novice game authors to specify a specific game object such as a chest, door or creature as this is equivalent to a single variable assignment. However, we have identified that dynamic assignments of game objects is a challenge for authors who do not have programming experience. For example, imagine the following scenario:

> You enter a room and the door locks behind you. Inside the room there is a troll and five chests. You are not able to defeat the troll, so you quickly look through the chests until, in the fourth one, you find a magic sword allowing you to defeat the troll. Not long after, your character dies and you must restart from an earlier checkpoint. Again, you end up in the room with the troll. This time you go straight to the fourth chest, but the sword is not there. Now you must search all the chests to find the sword before you can kill the troll.

This type of scenario is common in games. Having the sword appear in a random chest during game play increases the replay value. But, in order to do this, *what* chest to place the sword cannot be pre-specified and instead must be decided during game play. Other examples of dynamic decisions made during game play include having the creature closest to a door open it, or deciding where to place extra health and ammo in a first person shooter game.

To create a dynamic assignment, authors often need to link together multiple dependent varaibles, a task which is difficult for non-programers. Unfortunately, authors who avoid dynamic assignments have limited ability to determine and act on information discovered during game play. We present a new method to create dynamic object identification that we call descriptions.

2. DESCRIPTIONS

Most game scripting languages contain a large set of definitions (usually as function calls) that an author can use

FDG'11, June 29-July 1, Bordeaux, France.

to get information about the game. Example calls are "Get Nearest Creature" or "Toggle Door." Each call usually has a set of parameters to be set, such as the nearest creature to *what* or *what door* to toggle. By using these calls, and often linking multiple ones together, an author can define a binding to a dynamic game object.

Descriptions are a new method we created to allow authors to build these dynamic variable assignments without having to write any code. They are built on top of a scripting language. Authors *describe* the dynamic variable assignment they want as a plain english sentence, and the scripting code is then generated from the description. Descriptions are built upon three main ideas: variable context, a graph abstraction, and backward chaining.

First, a description requires knowledge of the variable it is going to set, specifically the type (creature, door, object, etc) of the variable. This knowledge can be used to filter and only show authors applicable definitions. That is, if the author is describing a door, it will not show descriptions that return a creature. While programmers are use to matching return types, non-programmers are not.

The second idea is based on abstraction. Knowing the intended type of a description reduces what definitions the author can choose from. However, in most cases the resulting list is still much longer than desired. Fortunately, we discovered that multiple definitions could be grouped together to create an acyclic graph, where each path through the graph represents a different definition. An example of a graph is shown in Figure 1. This graph representation dramatically reduces the number of descriptions that need to be shown to the author without decreasing expressibility. In addition, graph creation sometimes reveals useful descriptions that may be initially missing as individual definitions, so expressibility can actually be enhanced. Each graph is constructed so that the descriptions it contains can be read as sentences, making the descriptions easier to understand. The graphs are made up of start nodes (no incoming arrows), end nodes (either no outgoing arrows or all outgoing arrows go to optional nodes), red option nodes (need to be set by the author), green optional nodes (nodes that can be included to generalize the description), and black text nodes (to provide structure to the resulting sentence created by the graph). Each path through a graph will produce a different game script.

Using the graph, the author chooses a category of descriptions to work with, instead of choosing a specific definition. In Figure 1, the default description is "*Find nearest creature to object*". Given the default description, the author is able to modify it by adding or changing the description to correspond to any path through the graph. The author can edit this description using a small number of structured editing operations based on the internal graph representation.

The third idea involves the mechanism by which a series of dependent variable assignments is constructed. The variable assignment authoring process uses forward chaining while descriptions use backward chaining. For example, consider a situation where the author wants to award 100 gold pieces (GP) for each level that the PC has, rather than giving a fixed reward. Assume there is an action that can assign an arbitrary number of GP with the number as an option: *Assign <amount> GP to <recipient>*. With definitions, the author's goal is to create a definition for 100*(The PC's total levels) and use this definition as the amount. This

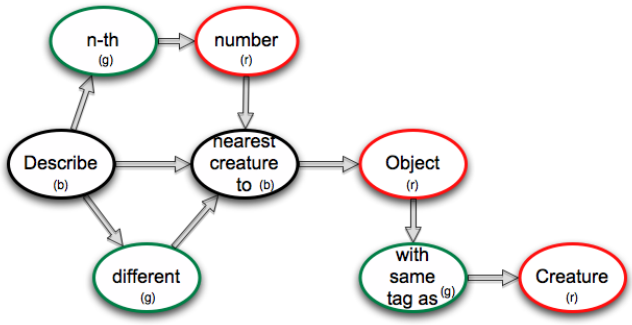

Figure 1: Example graph representation of the "Nearest Creature" graph with the five original definitions that were used to create the graph. However, the graph can be used to create six different definitions. The color of each node is also marked with a "b" (black), "r" (red) or "g" (green) for viewing in grayscale.

requires two definitions. The first defines the player's total levels and the second multiplies this number by 100 to determine the amount of GP to award. This is a forward chaining process: *Define* ***The Total Level*** *as <Creature>'s total level*, with the *<Creature>* option set to "PC" and then *Define* ***Product*** *as <number1> times <number2>*, with the *<number1>* option set to "100" and the *<number2>* option set to "The Total Level". This order is necessary since the second variable assignment cannot be completed until the first variable assignment exists. However, this order is not very intuitive for non-programmers, since the author must reason abstractly from the assign GP action all the way back to the first definition without authoring anything.

With descriptions, the author starts with the action: *Assign <amount> GP to <recipient>*. Then, by selecting the goal option: <amount>, the author launches a describer that can be used to construct a description for this option using backward chaining from the goal. As indicated earlier, since the describer "knows" that the author is setting a number, the describer can limit author choices to descriptions of numbers. In this case, the author selects a description category (math) that multiplies two numbers together. The author can set one of the numbers to to the constant 100. Next, the author can decide to further describe the other number. This creates another description in the describer, where the description type is again number. This time the author selects a different description category (Creature Statistics), sets the creature to the PC and specifies the statistic as the creatures level.

We believe that authors will prefer to use goal-driven descriptions to definitions that can be created with no context and that authors will be more effective using descriptions. To test this hypothesis, we conducted a user study.

3. METHODOLOGY AND RESULTS

We conducted a user study to evaluate our new method of descriptions. To conduct this experiment, we implemented descriptions in ScriptEase. ScriptEase [8, 5] is an ongoing project at the University of Alberta aimed at creating a tool to allow non-programmers to create their own video game stories without having to learn to program. Currently, ScriptEase allows non-programmers to create game modules for *Neverwinter Nights* [1]. ScriptEase contains a large generalized pattern catalogue of common events and actions that occur in video games. Authors choose generic patterns and then customize them for their story by setting options (such as which game objects to use in the pattern) and by adding and removing actions. Once all options have been set, ScriptEase generates the scripting code. The original method in ScriptEase for selecting dynamic game objects was through choosing and linking a series of definitions.

Our primary goal was to discover which method the participants preferred and our secondary goal was to determine which method was more efficient. Each participant was given two different test sets (A and B) and used one method (descriptions or the original definitions) on each set. Half of the participants used test set A with descriptions while the other half used B. In addition, half of the participants started with descriptions and the other half with definitions. Each test set contained 5 statements, each specifying a series of variable assignments to complete the statement.

The participants were undergraduate students in the University of Alberta's Psychology 104/105 classes in the Winter 2009 semester. They were 18 to 22 years old (mean and median: 19), 1st to 3rd year (mean: 1.7, median: 1) and received course credit for participating. We had 49 complete data records representing 33 females and 16 males.

A 3-factor ANOVA test was run to compare the main effects and interaction effects of the order (definitions or descriptions first), method (descriptions or definitions), and test set (A or B) on the number of statements the participants were able to complete. All three factors produced statistically significant main effects at a confidence level of 99%. The method result (p=2.84e-4) provides evidence that the students completed significantly more statements using descriptions than using definitions. The results also show that order (p=6.93e-8) was important; the students completed more statements using their second method which implies that learning took place from one method to the other. The study also showed that the Test Set (p=6.58e-3) influenced the results; students were able to complete more statements in Test Set B than Test Set A. Our hypothesis that descriptions are more efficient is true based on the greater number of statements the participants completed using descriptions as compared to definitions.

None of the interaction effects (pairs of effects) were significant at the 99% confidence level. The participants completed an almost identical number of statements using definitions regardless of which Test Set (A=1.08 vs B=1.00) they were working on and whether definitions were used first or second (1st=1.00 vs 2nd=1.07). However, the participants were able to complete more statements in Test Set B than Test Set A when using descriptions (A=1.42 vs B=2.42), but the order had less of an impact (1st=1.77 vs 2nd=2.09).

We also had the participants fill out a survey where they were asked which method was easier, faster, more intuitive, and better overall. We tested whether descriptions were better at a 99% confidence level for each aspect. And the results indicate that descriptions are faster (p=0.0061) and better overall (p=0.00082) but not significantly easier (p=0.011) or more intuitive (p=0.077). Our hypothesis that descriptions are preferred is true based on the better overall survey result.

4. CONCLUSION

There is a rise in the number of individuals who want to author their own video games. This trend is increasing the demand for more tools to aid non-programmers in authoring their own games. Tools like ScriptEase and Kodu, that are focused on game creation, Alice, which can be used to create animated stories and Scratch, which simplifies programming can all be used to lower the entry bar for prospective game authors.

This paper has revealed an important impediment to game authorship, the difficulty of specifying a game object whose identity is not known until game time. We have proposed an alternative solution to the current use of a series of variable declarations that we call descriptions. Descriptions aid the author by limiting the number of selections to only those that describe game objects with the correct type for the given context. A user study showed that authors were more efficient when using descriptions. Since the results of the user study were so promising, ScriptEase 2, which is currently in development, will replace definitions with descriptions. Perhaps other similar tools should observe the lessons of this user study as well and consider replacing free format forward chaining series of components by context sensitive backward chaining components that can reduce the number of inapplicable components the author must select from.

5. ACKNOWLEDGMENTS

The financial support of Alberta's Informatics Circle of Research Excellence (iCORE), Canada's Natural Sciences and Engineering Research Council (NSERC) and the Canadian Graphics, Animation and New Media Networks of Centres of Excellence (GRAND-NCE) is greatly appreciated. We also acknowledge the great work of the entire ScriptEase team, past and present.

6. REFERENCES

[1] BioWare. Neverwinter nights. Website, 2009. `http://nwn.bioware.com/`.

[2] CMU. Alice. Website, 2009. `http://www.alice.org`.

[3] CMU. Storytelling alice. Website, 2009. `http://www.alice.org/kelleher/storytelling/index.html`.

[4] B. S. LLC. The elder scrolls. Website, 2011. `http://www.elderscrolls.com/`.

[5] M. McNaughton, M. Cutimisu, D. Szafron, J. Schaeffer, J. Redford, and D. Parker. Scriptease: Generative design patterns for computer role-playing games. In *19th IEEE International Conference on Automated Software Engineering (ASE)*, pages 88–99, 2004.

[6] MIT. Scratch. Website, 2009. `http://scratch.mit.edu`.

[7] M. Research. Kodu. Website, 2009. `http://research.microsoft.com/en-us/projects/kodu/`.

[8] ScriptEase. Scriptease. Website, 2010. http://webdocs.cs.ualberta.ca/ script/.