Programming without Code: A Work in-Progress Paper

Catharine L. Brand
Computer Science Department
University of Colorado, Boulder
bb@indra.com

Keywords: POP-II.A. novices, POP-IV.B. user interfaces

Abstract

This paper briefly describes a natural language interface for educational games that enables children to create and modify objects in a game world.

Introduction

Can children modify and extend software without programming? My project focuses on creating an interface for educational games that would permit players without programming experience to create new objects by using a natural language description of the game to copy and modify them.

Upper elementary and middle school children spend much of their free time playing computer games. Socializing often involves working on a game in groups, either showing off scenarios to other players or playing against each other. Building layouts of objects such as cities containing well-known landmarks or setting up battle sequences draw many children into participating in a game. They trade advice, critique each other's work and learn a tremendous amount about how a particular game works. Along with many researchers (e.g. Games to Teach, http://www.educationarcade.org/gtt/), I have wondered whether some of this energy could be directed towards educational software.

Related Work

My work builds upon the ideas of others. In the PUPS system, Anderson and Thompson (1986) introduced a type of analogical generalization based on substitution informed by annotations, which provided necessary background knowledge. In Aspect Oriented Programming, (Kiczales et al.,1997) software designers separate the different components and aspects of a piece of software and then write a special module which generates executable code by automatically recombining them. Andri Ioannidou, a recent University of Colorado Ph.D., created the Programmorphosis system, which supports end-user programming at a very high level. The Programmorphosis wizard enables the rapid creation of simulations but does not aid in making changes to a simulation once it has been created. (http://www.cs.colorado.edu/events/defenses/2002-2003/ioannidou.html)

My experience with EcoWorlds, a system created by Cyndi Rader, Gina Cherry and myself (Rader et al. 1998, Brand et al. 1998), has also influenced my current work. We designed EcoWorlds as a software environment to support the creation of ecosystem models by elementary school students. EcoWorlds contained natural language templates for creating plants and animals. It was built as a layer on top of the visual programming language AgentSheets (Repenning, A. 2000, http://agentsheets.com/).

For several years, we worked with fifth grade classes, refining the software and developing a related curriculum. The fifth graders enjoyed the visual aspect of the program, especially drawing their own creatures and showing the characteristics they had chosen for them. They spent hours building elaborate environments with unique layouts of land, water, plants and animals. Despite our visual programming environment designed for simplicity and ease of use, students struggled to program their animals' behavior. They could modify the description of an animal in a template but most could not understand the how the underlying rule-based system worked and why their plants and animals behaved as they did.

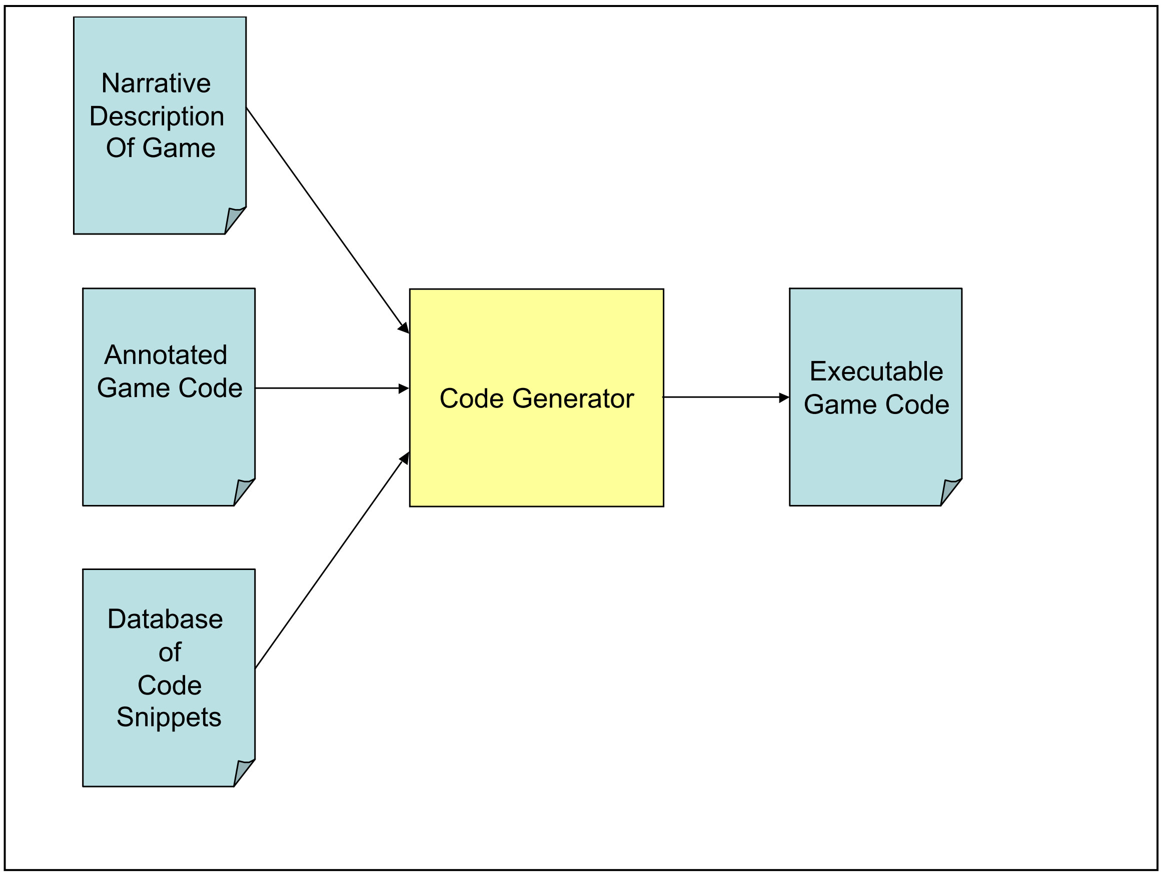


Figure 1: system overview

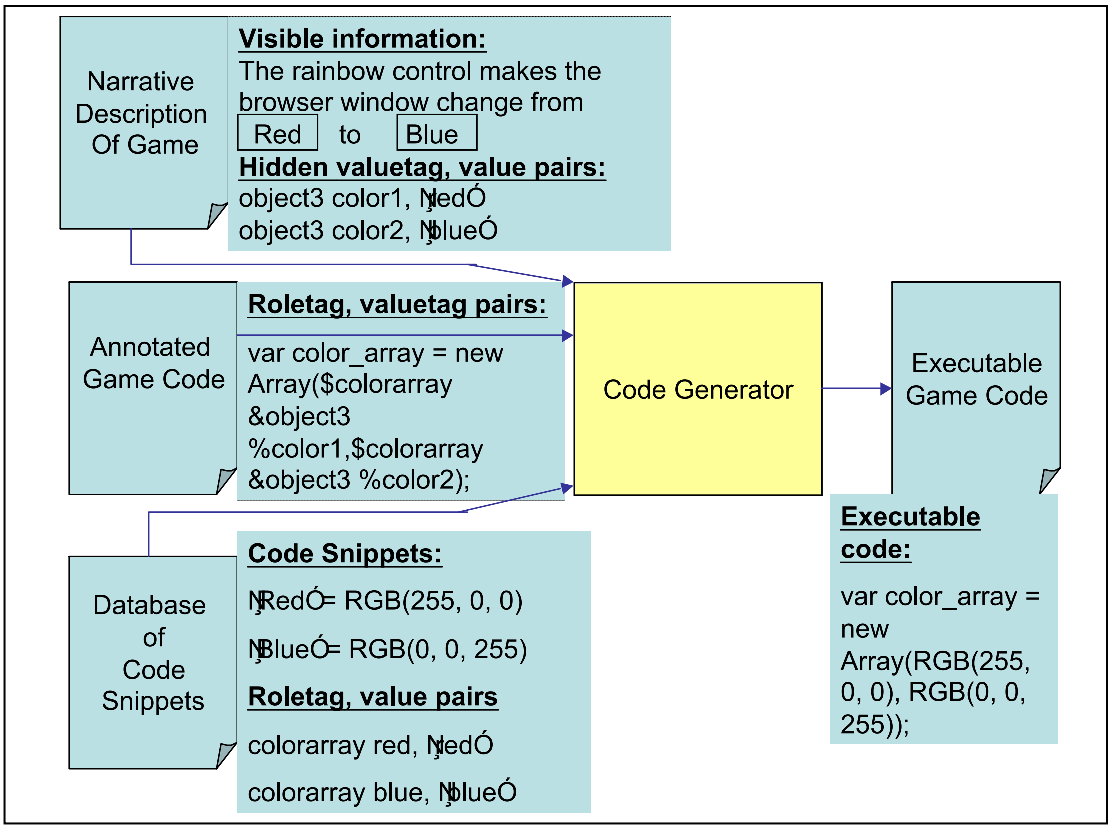
Objectives

My goal is to create a simple, accessible interface for educational games that would let non-programmers easily modify and create objects within the game framework. This interface would be particularly appropriate for children aged 10 to 12, who are old enough to enjoy creating and experimenting within a simulated world but too young to have taken formal programming classes yet. The interface should permit changes at a fairly detailed level so that children could create objects and scenarios that they envisioned rather than being limited to those planned by the game designers. Making modification easy should support tinkering and experimentation. I also wanted a system that could provide more alternatives as users add information.

Proposed solution

My programming without code system contains several parts (Figure1): a natural language narrative that describes the objects in the game, an annotated copy of the code to be generated, a database of information used to generate code and a code generation module. The discussion below will describe a simple example in which users can modify a description of web browser controls and then create their own version of a web browser with functional buttons.

Figure 2: code example



The rainbow control makes the browser window change from Red to Blue

Hidden valuetag, value pairs:

```
object3 color1, ŅRedÓ
object3 color2, ŅBlueÓ
```

```
var color_array = new
Array($colorarray
&object3
%color1,$colorarray
&object3 %color2);
```

```
ŅRedÓ= RGB(255, 0, 0)
ŅBlueÓ= RGB(0, 0, 255)
```

```
colorarray red, ŅRedÓ
colorarray blue, ŅBlueÓ
```

```
var color_array =
new
Array(RGB(255,
0, 0), RGB(0, 0,
255));
```

The player interacts with this software by editing a natural language description of the mutable objects in a game. This narrative is designed to highlight interesting changes to the game objects. The features of an object that may be changed by the player appear in textboxes with default values filled in. At the moment, the user may freely enter any value into the textboxes in the description. In the near future, feedback from the system will encourage the user to provide missing code or make another choice if the database does not contain the information necessary to generate useable code for a particular value.

Figure 2 illustrates the narrative describing a single control in my user-modifiable web browser example. In my software, the valuetags are the names associated with each user-supplied value, e.g. 'object3 color1' is the valuetag for the value 'red'. The user cannot see the valuetags but they give the program a way to uniquely identify each value and to link that value to some code.

My system requires an annotated version of the program code for the game that will be generated based on the player's input. This code must be marked with roletag, valuetag pairs wherever user-supplied information will determine the characteristics of an object in the generated program. The roletag identifies a particular point in this code. The valuetag marks a place into which any one of a group of values associated with that valuetag may be placed.

The software also contains a database of information needed to generate runnable code. Roletag, value pairs are matched with either snippets of actual code or pointers to code snippets as in Figure 2. These code snippets can be plugged into the annotated program by the code generator. This database could grow over time if a system administrator or knowledgeable user added more values together with their associated code to it.

When the user has completed her changes to the narrative description in the browser window, she invokes the code generator by clicking on a button in the same window. The code generator reads through the marked up code for the game. For each roletag, valuetag pair in the code, the code generator retrieves the value supplied by the user. Then for each mutable feature in each object, the code generator retrieves the code snippet associated with the roletag, value pair and substitutes it for the roletag, valuetag pair in the code. When all the substitutions have been completed, the code generator creates a JavaScript file that can be loaded into a web browser window to produce an interactive game.

Players will easily be able to create new objects in this system. The narrative description of the game objects contains flags that tell the player that segments of the description may be copied to create a similar object. When the user clicks on a copy button associated with an object, a description of the new object appears, complete with boxes for entering its values. By entering different values into the boxes, the user may create an object that is similar to the original but has its own behavior. For example, a web browser control that makes the browser go back to it's previous URL could be copied and modified to produce a control that makes the browser go forward by changing the appearance of the control and by changing the name of the behavior assigned to it.

I believe that this system will permit players to create objects that elaborate in interesting ways on the basic structure that I provide. I plan to extend the copying facility so that parts of objects may be copied and modified as well as entire objects. In the case of the rainbow browser control in Figure 2, the user should be able to change the description from " The Rainbow control makes the browser window change from red to blue" to "The Rainbow control makes the browser window change from red to blue to yellow". This software will enable modification at a very detailed level without requiring extensive programming knowledge. I hope that children will find it easy to say, "I want something like that thing only I want to change this value."

Next Steps

This approach to programming without code still needs extensive testing and development to prove its utility. My first step will be to design and code a more game-like example. I believe that some science topics can be made as exciting for children as fantasy games. I am particularly interested in creating a game that would help children explore the human immune system. This subject offers the advantage of a good guys against the bad guys conflict as components of the human immune system struggle to overcome or disable invading bacteria and viruses. I think that I can code a series of behaviors for immune cells and provide a narrative description that would encourage children to build a variety of more or less realistic immune cells. I plan to store a rich set of values and code snippets in the database to permit as much flexibility as possible for the user creations.

In addition to providing an appealing learning environment with a game-like flavor, I plan to push on improving the software beyond its current rather crude stage. At the moment, I am rewriting the existing JavaScript code generator in Python so that it is easier for the software to process all the objects in the narrative description and generate a new version of an interactive game. I would also like to provide a tool for a teacher or more knowledgeable user that would make it easy to create or edit the narrative description of game objects. I would also like to provide a simple method for a user with some programming experience to add code to the database, probably some combination of a dialog box and a quick check that the snippet is a piece of well-formed code.

Testing also forms an important piece in my plans. I hope to work with 10 year olds from a local elementary school. Initial testing will focus on ensuring that the system is comprehensible to and usable by children in this age group. Inevitably, user testing brings up issues overlooked by the designer. After whatever redesign seems necessary to increase usability, I will focus my testing on whether the children actually learn anything about the human immune system after using the software.

Comparing pre- and post-interviews on this topic will give me a method for understanding the potential of programming without code as a learning environment for children.

If adequate funding can be obtained, Professor Lewis and I have some ideas for elaborating on the basic idea of programming without code. We would like to create a web site that would support children in creating simulations of history. For example, in addition to writing a report on the gold rush and the mining camps it spawned in Colorado, students could invent characters and give them appropriate behavior through a programming without code type interface to a game engine.

Acknowledgements

Thank you to Clayton Lewis for his help in designing this software.

References

Anderson, J.R. and Thompson, R. (1986) Use of analogy in a production system architecture. Paper presented at the Illinois Workshop on similarity and Analogy, Champaign-Urbanan, June, 1986.

Brand, C., Rader, C., Carlone, H. and Lewis, C. (1998) Prospects and challenges for children creating science models. paper presented at the National Association for Research in Science Teaching (NARST) 1998 Annual Meeting, San Diego, CA, April 1998.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.-M. and Irwin, J. (1997) Aspect-oriented programming. *Proceedings of the European Conference on Object-Oriented Programming(ECOOP*), Finland. Springer-Verlag LNCS 1241. June 1997.

Rader, C., Cherry, G., Brand, C., Repenning, A. and Lewis, C. (1998) Designing mixed textual and iconic programming languages for novice users. *Proceedings of 1998 IEEE Symposium on Visual Languages*, Halifax, Nova Scotia, IEEE Computer

Repenning, A. (2000) AgentSheets®: an interactive simulation environment with end-user programmable agents. *Interaction 2000*, Tokyo, Japan. Paper downloadable at http://agentsheets.com/about_us/Papers.html.