AN EVALUATION OF INTERACTIVE CURRICULUM USING THE JAVA INSTRUCTIONAL GAMING ENGINE

By

JAMES V. VAN BOXTEL

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Engineering and Computer Science

MAY 2010

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of JAMES V. VAN BOXTEL find it satisfactory and recommend that it be accepted.

Scott A. Wallace, Ph.D., Chair

Wayne O. Cochran, Ph.D.

Orest J. Pilskalns, Ph.D.

ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation under Grant Numbers DUE-0633726 and CNS-0829651. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the NSF.

AN EVALUATION OF INTERACTIVE CURRICULUM USING THE JAVA INSTRUCTIONAL GAMING ENGINE

Abstract

by James V. Van Boxtel, M.S.
Washington State University
May 2010

Chair: Scott A. Wallace

This paper focuses on using interactive curriculum to engage students and ensure they learn the course materials in computer science classes. We review previous work in interactive curriculum, and compare two teaching frameworks to the Java Instructional Gaming project. We then present two interactive assignments we created and evaluate there effectiveness through direct assessment and surveys. Finally we discuss previous work in evaluating API design and present a tool we created to help improve the JIG API.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

Dedication

This thesis is dedicated to my beautiful fiancée Tessa, my parents, my friends for all their love and support, and most of all, to my God who offers redemption to all and gives guidance and everlasting life daily.

I would also like to thank all my professors for their support, advise, and entertaining education. Finally, thanks to all the staff for making Washington State University Vancouver such an amazing university.

CHAPTER ONE

INTRODUCTION

How can we engage students and ensure they learn the course material in computer science classes? This question is arguably one of the fundamental questions of computer science curriculum research, but there are many different ideas of how to answer it. This question is extremely relevant to Computer Science, as attrition rates are so high. Theresa Beaubouef reports that attrition rate are as high as 30 to 40 percent in some institutions, noting that most of the drops are in the first and second years of Computer Science work [2].

Interactive curriculum has emerged as a way to help fight this problem. It is a type of curriculum that uses graphical results and a game-like environment to help engage students. A variety of educators have used interactive curriculum in their classrooms [21] and found that this type of curriculum is, for many students, enjoyable and motivating. Uses range from: learning through playing a game [10]; implementing a single part in an existing game [25]; and creating a full game from start to finish [16].

The Java Instructional Gaming Engine was developed to support interactive curriculum [24]. It is a java-based 2D engine that simplifies the process of creating 2D games and game-based curriculum. The engine provides sound playback, graphics manipulation, and a game loop along with supporting classes.

In this paper we have two objectives. Foremost, we intend to illustrate how the JIG engine can be used effectively in an undergraduate classroom setting and discuss some of the lessons learned in creating curriculum. Secondly, we discuss the evolution of JIG itself along with a tool and general approach that may help improve libraries like JIG.

To show how JIG can be used in the classroom effectively, we developed two interactive assignments and tested them in the classroom. The first assignment is based on simulating how ants collectively behave. Students implement a small portion of code that tells the ant

how to interact with the world. In the second assignment, students implement an abstract list data type that is used as the backbone for a puzzle game. We hypothesize that these assignments will increase student engagement and maintain learning when compared with traditional assignments. We use student surveys before and after the curriculum to gauge student perceptions and learning. In addition, we evaluate the success of the learning objectives by direct assessment. We discuss what worked well in our studies, and what did not.

To improve JIG, we developed a tool to help identify what elements of an API may need improvement. The tool consists of an Eclipse plug-in that can create a list of all the methods used in a library along with the number of times they are referenced by client code. We hypothesize that less-used methods are more likely to need refactoring or improvement. By comparing reference count to a manual classification of methods that require improvement, we then show that reference count is indeed a useful heuristic for identifying aspects of a library that may benefit from modification.

Chapter 2 starts by documenting other interactive curriculum and frameworks and compares them with the JIG project. Next, we discuss the curriculum we have created and evaluate it in the classroom. In Chapter 3, we discuss previous work in API design and ways of predicting problems in APIs. We then discuss our tool and its results when applied to the JIG API. Finally we conclude with our observations on interactive curriculum and the evolution of JIG.

CHAPTER TWO

INTERACTIVE CURRICULUM

In this section we discuss the current state of interactive curriculum and classify it. We then describe some of the leading curriculum repositories. Finally, we look at some popular teaching frameworks and compare them with JIG.

2.1 Curriculum Categories

The use of interactive game-based curriculum has taken off in recent years. Most curriculum can be put in one of four main categories. Sometimes curriculum span multiple categories, however it is useful to understand the benefits of each category type. Kelvin Sung gives a similar break down of curriculum into categories in his paper [20].

2.1.1 Edutainment: Learning through play

Curriculum in this category involve learning through playing games that involve the subject being taught. These games have the learning experience built directly into the game. No programming or logic is done outside the game, instead the student learns by interacting with the world.

One way this is done is through games specifically designed to teach the subject. In Ray Giguette's paper, the author presents games that allow beginning computer science students to start thinking about algorithms as strategies for solving a problem [10]. For example, a student is given a maze to navigate in which they must find the shortest path while dealing with booby-traps. Thus, the student can think about their strategy (essentially an algorithm) without actually doing any programming. A screen-shot of the game is given in Figure 2.1.

Another example is in Tiffany Barnes Game2Learn program [1]. In one example, the student helps fix a local egg factory owner's machine in a game. Although they are called machines in the game, they are actually algorithms disguised as an object in the game

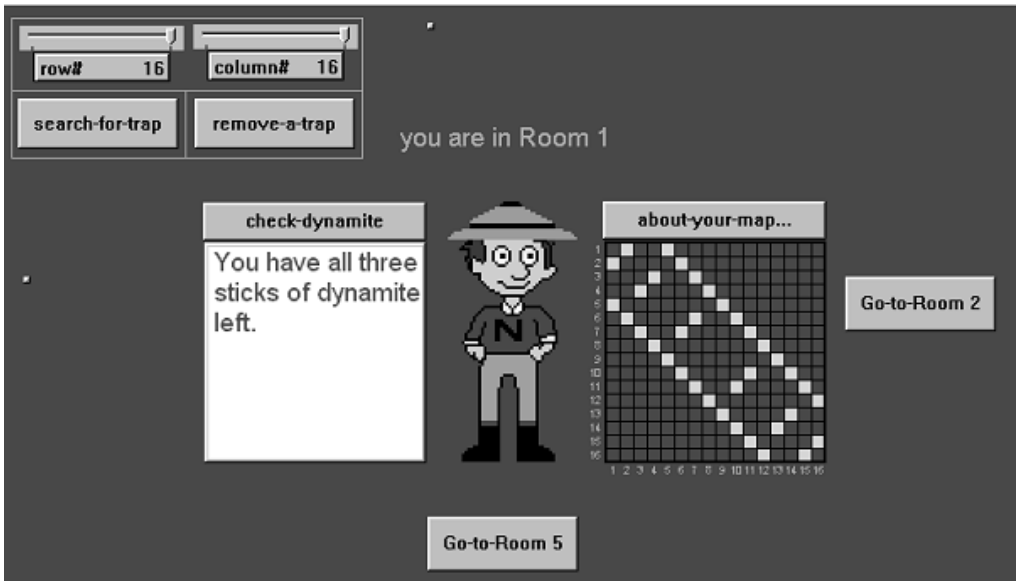

Figure 2.1: *Ray Giguette's edutainment game where the student plans how they will navigate the maze, learning a strategy (algorithm) [10]*

world. The student answers multiple choice questions to fix the machine. This particular machine is a nested `for` loop designed to drop eggs into crates, but it has a problem. It drops too many eggs because the `for` loops start at zero. After fixing the problem, the solution is reinforced as the player's character explains the solution to the machine's owner. An example of the game is given in figure 2.2.

Another way of creating edutainment is re-purposing consumer games to teach students. An example can be seen in an after school program where students played *Civilizations III* [19]. In this example students played the historical game after school and were quizzed on topics related to the game such as vocabulary and historical events. The goal of this program was to measure how students learn a strategy game in addition to how many historical facts students retained just from experiencing the game. Students were found to grow to an equal level of strategy when compared to the facilitators, but were not as good at articulating those strategies. As far as knowledge about facts, "each participant could place Egypt on a map, could give a general definition of the term hoplite and could identify

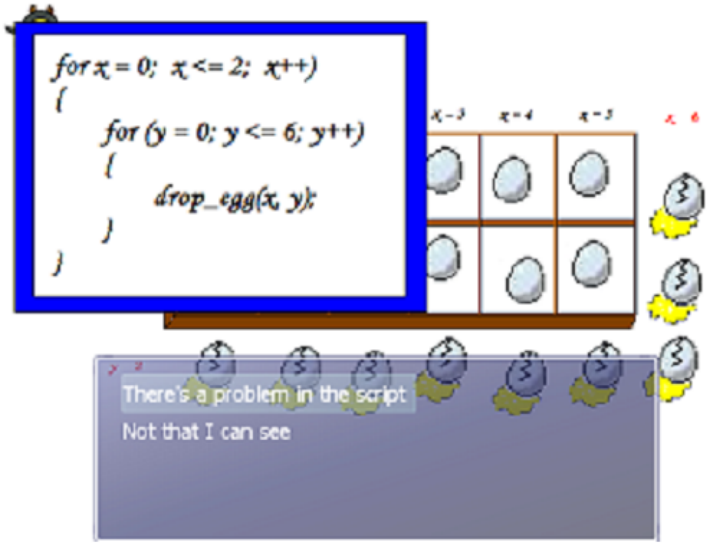

Figure 2.2: *Tiffany Barne's Edutainment game where the student examines the code for dropping eggs into crates and answers multiple choice questions to fix it. [1]*

five military units of the ancient civilizations" [19, page 40]. The more proficient players also "perfectly matched quite a few historical terms to their respective definitions, terms which included monarchy, hoplites, galleons, barracks, Babylonia, Thebes, Great Library, despotism, cathedral, Colossus, iron working, aqueduct and war chariot." [19, page 40]

2.1.2 Plug-N-Play: Plugging in the final piece

This category typically involves environments that are mostly complete, but require the student to fill in one small piece that will complete it. Typically programming this correctly allows the player to win, or see the environment function properly [11].

This category has been used successfully to teach Artificial Intelligence algorithms such as Djikstra's algorithm [25]. In Scott Wallace's Introduction to Artificial Intelligence class, students were presented with a overhead world and bugs that must go from any point on the tile-based map to the exit. Students had to develop their own implementation of Djikstra's algorithm in order to tell the bugs which way to go. After students developed their solution,

Figure 2.3: *Scott Wallace's game where the students program a way to calculate the distance to the exit from any tile. The bugs then know which path is the shortest to traverse. [25]*

the game automatically loaded it and showed the routes graphically on the screen (Figure 2.3). In a later assignment, students implemented the A* algorithm so that they could control a robot on screen and tell it to go from any spot on the map to any other spot in the most efficient way (different tiles had different speeds).

In Huang's paper on strategy games, he discusses how the game Connect 4 can be used to teach a search algorithm, minimax [14]. Connect 4 is a game where students drop checker pieces into vertical columns and try to get four of their pieces in a row. The players take turns and pieces stack on top of each other in the columns. In his class, students were given the skeleton code for the game that included everything except the strategy for the computer player. Students first implemented a computer player that did not use game-tree search. They then implemented minimax which is an algorithm that recursively looks at the best move for the player and his opponent, trying to maximize the player's reward, and minimize the opponent's reward. Huang reports that the students found the assignment both challenging and worthwhile and they also enjoyed the light hearted tournament they

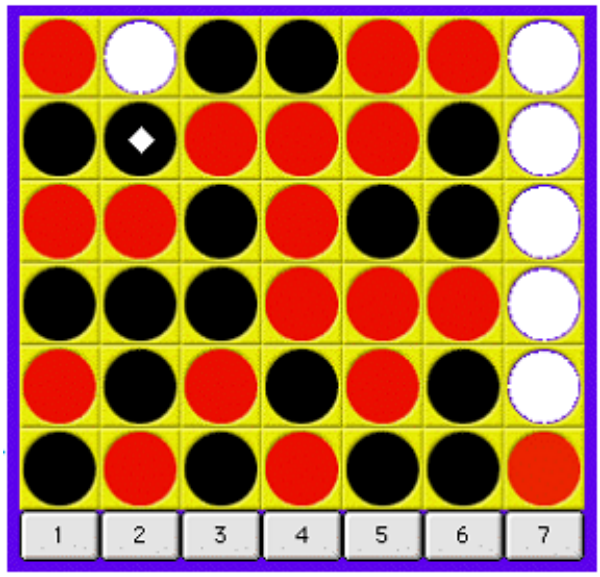

Figure 2.4: *Huang's version of Connect 4. Red is about to win by playing in the second column. [14]*

were able to do at the end of the assignment.

2.1.3 Game Design

In this category, students learn how to develop games. Classes that use this type of curriculum often devote longer time periods to each project since game development is time consuming. These student games often use a game engine or other framework as well, since it often makes the project load smaller.

One example of learning game design is Ian Parberry's senior capstone course at the University of North Texas [16]. Students created their own game from start to finish dealing with many of the different challenges such as graphics, sound, and physics. The goal of the university is to prepare students for game design jobs in the industry. To this end, they use C++ and DirectX and group the computer science students with a student artist. In addition, they bring in guest lecturers from the game industry and encourage the students to create their own game from scratch so they can learn the entire process.

Figure 2.5: *Jiménez-Peris's example image of the game Tetris [15].*

However not every game design class is focused on the game industry. Randolph Jones' paper discusses his teaching of a game design class where higher level design issues and object oriented design were emphasized over preparing students for the game industry. In this class, Java was used as the programming language even when it was in its early stages (Java 1.2) and was relatively slow. The author comments that he realizes this was a limitation but feels it served the purposes of the class. Over time Java continues to improve and this issue has become smaller over the years.

These two papers indicate that this category is typically presented to upper division students as it takes quite a bit of skill to design a full student game from start to finish. The classes in both the previously mentioned papers devote a significant amount of time to the final project, about half the semester. This should be taken into account when planning out course materials.

2.1.4 Small Game Implementation

Sometimes the instructor can limit the complexity of creating a game by providing a lot of the structure for the student. By providing the game rules, instructions on the graphics and some basic game loop code, lower level computer science students can still taste the rewards of implementing a game without the overhead of designing the game mechanics, graphics and sound.

For example, Jiménez-Peris's documents in his paper how they have taught using Asteroids, Tetris, Card Games, Image Compression, and more [15]. In the case of Tetris, the students are given the rules and some basic game code that allow them to read when keys

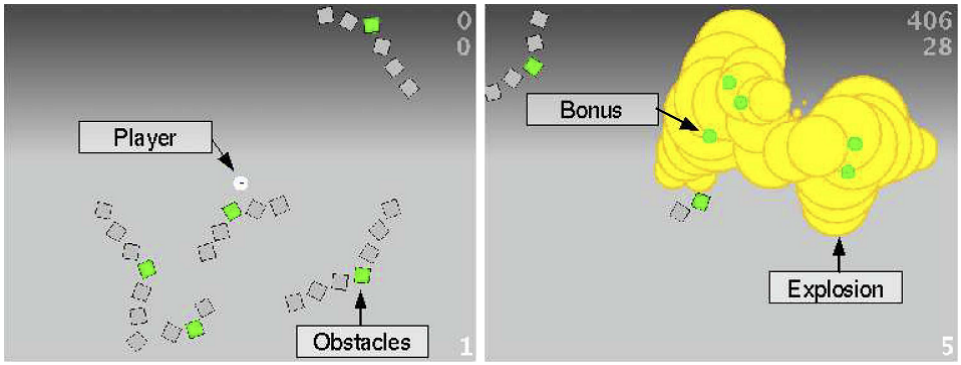

Figure 2.6: *Two screenshots from the game EEClone. The left one shows the player dodging the blocks, the right shows the player activating the explosion causing a chain reaction. [9]*

are pressed and track time. Tetris is a simple game where different sized blocks drop from the top of the screen and the player tries to stack them at the bottom in a way that they can fill each row in the screen completely. When a row is completely filled it is removed, and the other blocks are shifted down. Students can expand on the project as well by adding levels that increase the difficulty over time, making the blocks fall faster.

Katrin Becker's paper presents a list of games that are highly suitable as small game implementations [3]. She also details how students implemented the game Minesweeper at University of Calgary and were genuinely more excited about assignments then previous iterations of the class. She documents that one of the great things about implementing Minesweeper is just about every student has played it as it comes with Microsoft Windows.

2.1.5 Multiple Categories

Sometimes these categories are not so distinct. For example, at Ball State University a class taught design patterns throughout development of a game called EEClone [9]. In this game, the player controls a bomb and avoids the enemies until they decide to detonate and cause a chain reaction. The more explosions they cause, the more lives and points they earn. As the game progresses lives become harder to obtain and a timer runs out.

One of the patterns used in this game is the State pattern. In this pattern each state

an object can be in is represented by it's own object. This way, the behavior can be delegated to the state object rather than having `if` statements or `switch` statements scattered throughout the code. The Unified Modeling Language diagram given in the paper is shown to clarify in Figure 2.7.

In this curriculum, the students do get some leeway in adding features to the game, but the basic game is also set by the instructor and specific design patterns and topics are structured throughout the creation process to teach computer science subjects. This makes the project more of a merger of game design with a structured programming assignment than any one curriculum type.

2.2 Curriculum Repositories

In addition to the vast array of papers on the subject of interactive curriculum, there also exists various repositories of computer science curriculum.

One of the most famous of these repositories is the nifty assignments repository run by ACM's Special Interest Group on Computer Science Education (SIGCSE) [17]. This repository consists of six computer science assignments selected every year since 1999. Generally these assignments are of high quality and have an interesting aspect to them as they are peer reviewed.

The North West Distributed Computer Science Department (NWDCSD) is a project started to help educators collaborate and share each others curriculum in a online repository [23]. This project, unlike the Nifty Assignments project, is designed with discussion and collaboration deeply integrated with the repository.

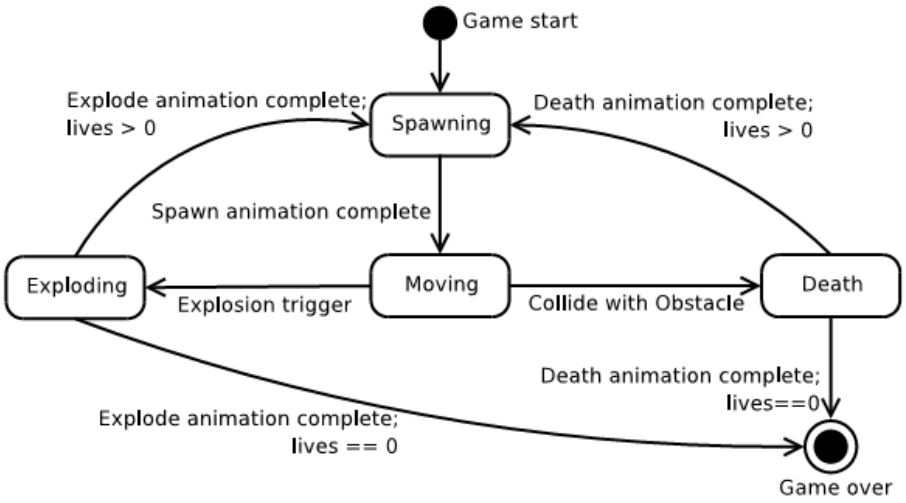

Fig. 2. State diagram for the EEClone player's sprite.

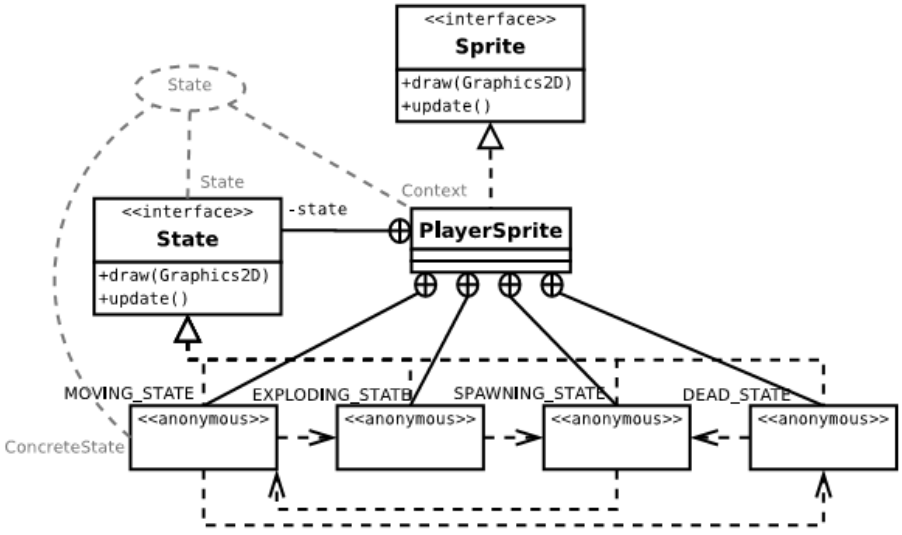

Figure 2.7: *UML Diagram showing the power of the State pattern. Taken from Ball State University's Paper [9]*

2.3 Curriculum Frameworks

A couple frameworks stand out as being particularly useful for teaching in the classroom. These frameworks implement much of the hard parts of interactive programs such as graphics, sound and interaction. They also often have various assignments already designed for them as examples.

2.3.1 Java Instructional Gaming Engine

The Java Instructional Gaming project was developed at Washington State University Vancouver and the University of Puget Sound with funding from the National Science Foundation. This project aims to provide a 2D Java game engine that is simple to use yet powerful. In addition, the project provides curriculum modules that instructors can use to teach students [24].

The engine was intended to be simple and easy to use in programming courses. It has a number of key attributes:

Java Java has quickly become one of the main programming languages taught at colleges. Java has also became the basis of the Advanced Placement exam for high school students [4]. This rise in beginning with Java has not gone without controversy [6], nevertheless it has became popular. As Java became one of the leading introductory programming languages, the need for a Java-based game engine increased. Because the engine is written in Java, many of the classes and data structures students typically need are built in as well. For example Lists, and Maps are built into the Java library and are optimized by the Java Virtual Machine.

2D Focus The JIG engine is developed with a 2D focus to reduce complexity and focus on design and programming skills rather than complex 3D modeling and other tasks that generally are limited to upper division courses.

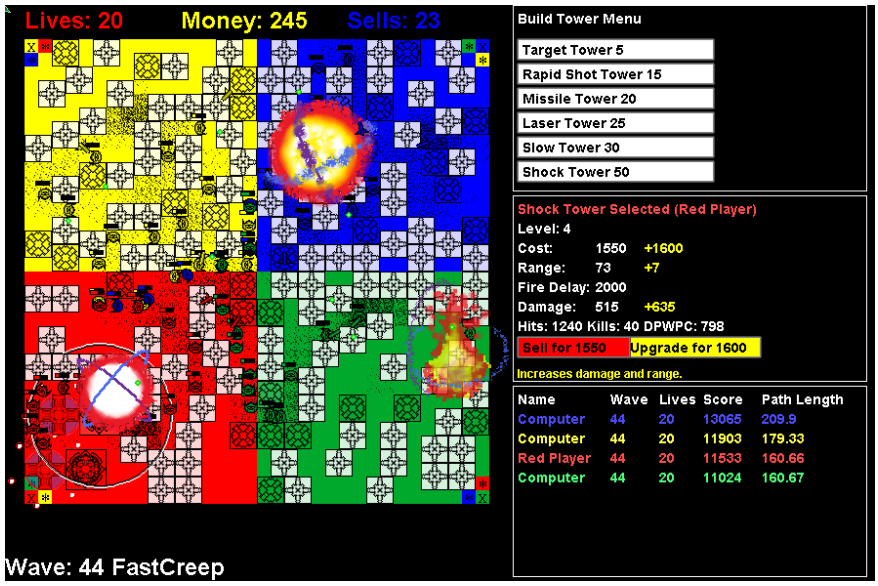

Figure 2.8: *A game that uses the Java Instructional Gaming Engine. This game takes advantage of alpha transparency that is built-in to JIG.*

Features The JIG engine contains a plethora of features, but it's modularity allows a small subset to be used in introductory courses. For example, a basic program might display an image on the screen and move it around based on keyboard input. Students can easily iterate on that design and add music, sound, transformed coordinate spaces, or fonts and alpha transparency using the provided classes.

Open Source Code Often times engines do not include the source code, or contain complicated build sequences that make understanding or altering the engine a nightmare. However due to the use of Java and all source code being included in JIG, changing the engine becomes much simpler. In addition, students are able to see how the engine works. This allows advanced users to go deeper in their studies if they desire.

Getting started with the JIG engine is relatively straight forward. Students add the JIG library to their project and create a subclass of the main game class. Users have the option between a Static Screen Game class or a Scrolling one, depending on if their game world

is bigger than the screen. Next the user creates objects that extend from the sprite class that will represent the elements in their game. They instantiate these objects in the main game class and respond to user input with their game logic. Users can use engine created images temporarily until they have their own. Typically users iterate from here and slowly add graphics, sound, text, and deeper game logic.

Users are encouraged to use a Integrated Development Environment when programming a game with JIG. This simplifies the compilation process and provides features like auto completion, error suggestions, and code formatting.

In the case of curriculum that uses JIG, often times the curriculum designer can create the module so students only have to implement one class to interact with the curriculum module. In this case the student compiles their class with the JIG jar included as a library. When the curriculum is run it automatically detects the student's compiled class file and loads it.

2.3.2 BlueJ

BlueJ first came out as a way to make Java programming easier and allow students to visualize class hierarchies. It is a fully contained environment that allows students to edit code, compile, and run all from a graphical interface.

One of the most powerful features of BlueJ is it allows you to create class hierarchies visually. For example, you can draw an arrow from one class to another, indicating it is a subclass, and BlueJ will automatically edit the code to make it a subclass. Additionally, the UML diagram will update when you change the code, ensuring your UML diagram is always up to date.

BlueJ is developed from the ground up as a way to introduce students to programming with objects first. Because it uses a integrated environment, students can run methods from the GUI without worrying about a main method. The GUI allows them to call a method

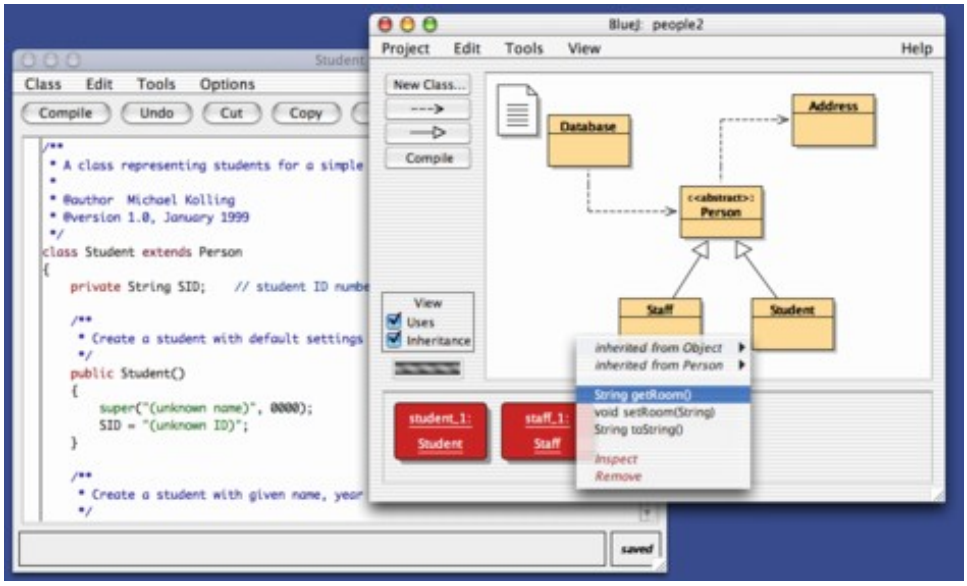

Figure 2.9: *The BlueJ environment.*

by mouse clicks and input boxes. The return value is then shown in a window. This makes students not need to worry about input and output when first programming as well.

These features help simplify beginning programming and let the students think about objects and functions first, before they have to deal with the syntax and special cases of the Java language.

2.3.3 GreenFoot

Greenfoot is a new environment that evolved from BlueJ. It makes visualizing simulations and games easier by providing a simple framework [13]. It is a self-contained environment just like BlueJ that allows easy editing of classes simply by double clicking them. However it also provides a graphical environment where scenarios can be created that students can interact with and program in visually.

Students load a scenario from one of the many examples or from the professor. Greenfoot then presents a world with objects and a map. The ant scenario is given as an example in Figure 2.10. When the simulation is run, it is automatically compiled. Students can

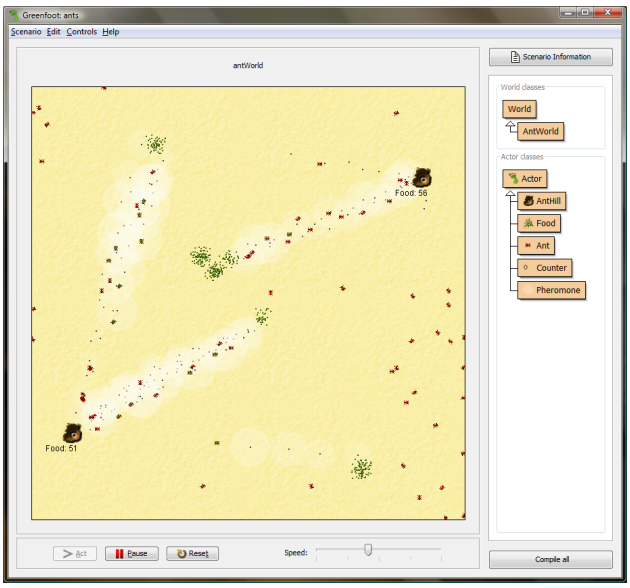

Figure 2.10: *A ant simulation provided by the Greenfoot team.*

control the speed of the simulation and pause or play it using the controls built into the environment.

Students can also manipulate objects in the world by moving them, or calling their methods via the right click menu. Adding new objects can be done by right clicking the class and picking the spot to place the object. Students can also inspect the properties of an object. This manual interaction allows the student to really experience the idea of objects and lets them experiment with less overhead then writing code to setup the world the way they want.

Students can also change the behavior of a object by simply double clicking the class and editing the code. This makes testing different behaviors simple and intuitive.

All of these features combined makes Greenfoot a great place to start for beginning programmers.

2.4 The Benefits of JIG

BlueJ and Greenfoot are mostly designed for introduction to programming courses. Thus they have some potential drawbacks when it comes to larger scale programming. It should be noted that while simplicity can be a drawback, it can also be a feature depending on the needs of the classroom.

BlueJ and Greenfoot would most likely not scale to a large project size due to the fact that it would be hard to view all the source files and navigate through them. The environments continue to improve with each version including features such as global search. However IDEs like eclipse have a much greater feature set for managing large projects.

In Greenfoot, the game or simulation must run inside the environment's window. While these windows help with interaction, debugging, and code design, they can also detract from the actual game once it is complete. Sometimes a game or interactive curriculum is most effective when the screen is not cluttered with menus and options around it. These windows can detract from the lesson being taught, or reduce the cinematic experience a game may be trying to create.

Another major drawback is the students do not get to use the features of a popular integrated development environment such as Eclipse. One of the really nice features of Eclipse is how it handles compile errors. Eclipse compiles code as you are typing and highlights errors as well as gives suggestions on how to fix them. These suggestions are often better then Greenfoot's suggestions. For example, I added a call to create a new Color object in both environments without importing the class first. In Greenfoot, I had to compile to see the error. I then got the error message "cannot find symbol - class Color". I could click the question mark button to get the following help description: "You are using a symbol here (a name for a variable, a method, or a class) that has not been declared in any visible scope. Check the spelling of that name - did you mistype it? Or did you forget

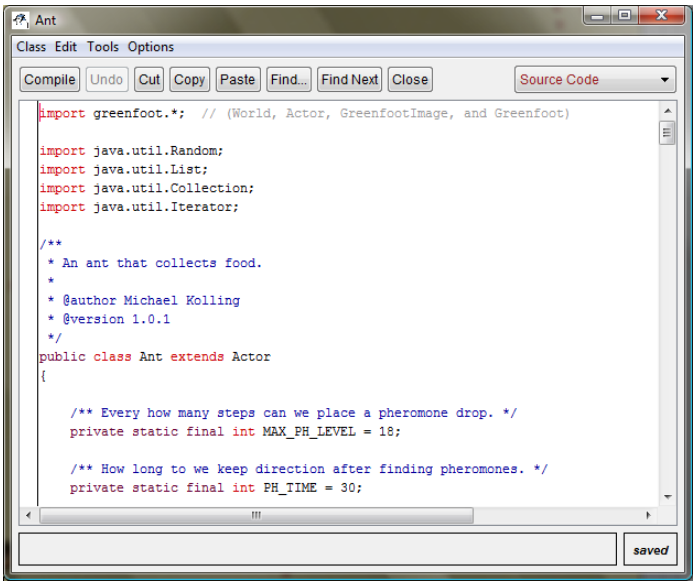

Figure 2.11: *Editing the Ant class using the Greenfoot editor.*

to declare it? Or maybe you did declare it, but it is not visible from here.". In Eclipse, I got the following error as soon as I had finished typing: "Color cannot be resolved to a type". A icon on the side then suggests I "import java.awt.Color;". By double clicking the suggestion, Eclipse automatically fixed my error.

Eclipse also provides a way to pop up documentation when you hover over a method. The pop-up shows the javadoc for the method, often alleviating the need to go look at the full documentation of the class. Eclipse also allows the user to jump to where a method was declared, or search for all the places it is used. Using a IDE like Eclipse also gets students used to tools that they will most likely use in the workplace.

With JIG, students can develop in the IDE of their choice, or from the command line if they so choose. Simulations can run at whatever screen resolution the programmer desires, even full screen. This makes JIG a good environment for someone making a slightly larger or more professional simulation or game. However, it does take a little more setup and programming knowledge. Specifically, the programmer must be familiar with the JIG

library so they can structure their program correctly and subclass the right classes. The programmer must also know how to setup a Eclipse project and include a library jar file, or know how to link using the command line compiler. Fortunately the JIG website explains these steps and has example games to help programmers learn how to use JIG.

CHAPTER THREE

THE ANT CURRICULUM

3.1 Ants Background

At Washington State University Vancouver, Professor Wayne Cochran has used ant simulation environments to teach a variety of different computer science topics. In a overhead world that is typically 2D, the ants start at their nest and explore the world scavenging for food. When they find food, they return to their nest dropping pheromones as they go. Slowly ants start to follow pheromone trails as the chemical increases.

This simulation can be implemented in many different ways but the ants typically have limited knowledge about their surroundings and have local or no communication with each other except through pheromones. In addition, the world contains boundaries the ants can not traverse.

In previous iterations of Professor Cochran's class, the ant simulation was implemented with text-based graphics. The simulation is programmed using the ncurses library to display the world. In this case, the world is grid-based and different text characters are displayed in the screen to indicate the different objects such as ants, food, the nest, and boundaries. Students are responsible for programming the entire world and ant behavior. It is up to the student to make sure the ants obey the world's laws. For example, ants should only move one square at a time in a 8-connected fashion and should only use information about the tiles immediately next to them. A screenshot of the text-based environment is given in Figure 3.1.

The goal of the text based assignment can vary based on the language used, but it usually involves giving the students some exposure to an object oriented language. Sometimes memory management like reference counting is also a learning objective depending on the

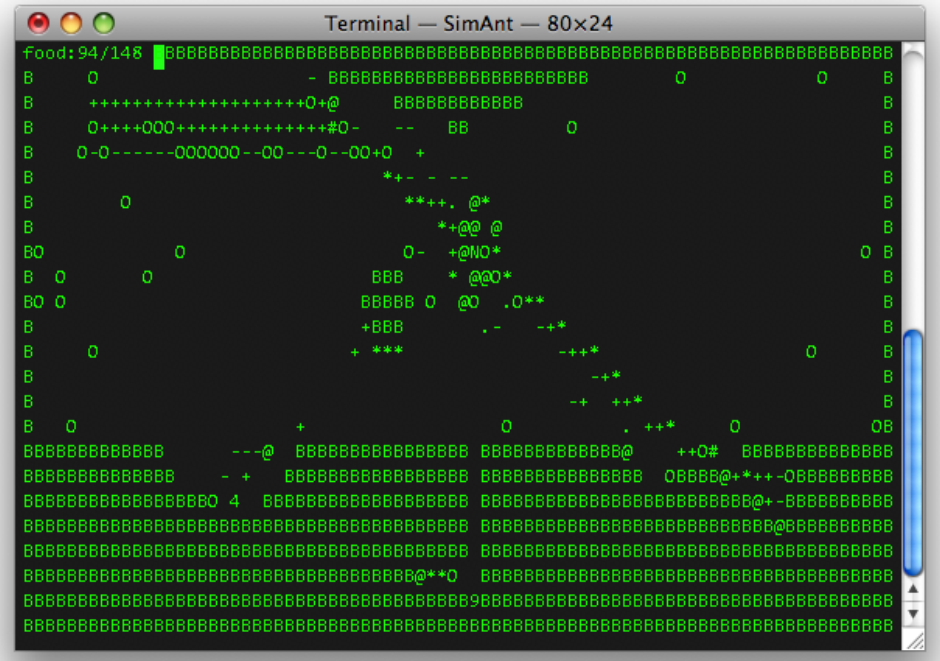

Figure 3.1: *A screenshot of the text based ant curriculum. O represents an ant, B represents a boundary, . - and + represent pheromones, and numbers represent the food left.*

language.

3.2 2009 Graphical Ant Simulation

The ant simulation created for JIG has a overhead view of the world like the text based simulation, but has much more immersive graphics. It uses the JIG engine which allows for images and zooming in and out. The result is a much more interactive experience and allows for larger scale worlds. Through in-game menus, the user can control the ant simulation. The user can change the level, pause or change the speed of the simulation, and select which ant solution to run so they can compare and compete ant solutions.

Since this version is written in Java, all of the parts of the environment are modeled with objects. Using a delegation pattern, the student only has to implement the ant's behavior. This reduces the complexity of the assignment since the student only needs to know about how the ant can interact with the world. Everything else is taken care of behind the scenes. Figure 3.2 shows the UML diagram for the ant curriculum. Students extend the AntBehavior class and override the update method which determines the behavior of the actual ant object. This way, students can only interact with the ant world using the methods in the AntControls object.

This curriculum is also "plug and play". Students merely compile their code in the correct folder and the curriculum automatically loads their solution and runs the ant simulation with it. Java is a dynamic language that allows classes to be loaded at runtime. This means the ant simulation framework does not need to be recompiled, it simply uses Java's dynamic class loading to load classes it finds in the solution directory.

We have two main goals for our ant simulation. First, we want students to perceive the curriculum as engaging and interesting. Second, we want our curriculum to teach object oriented techniques.

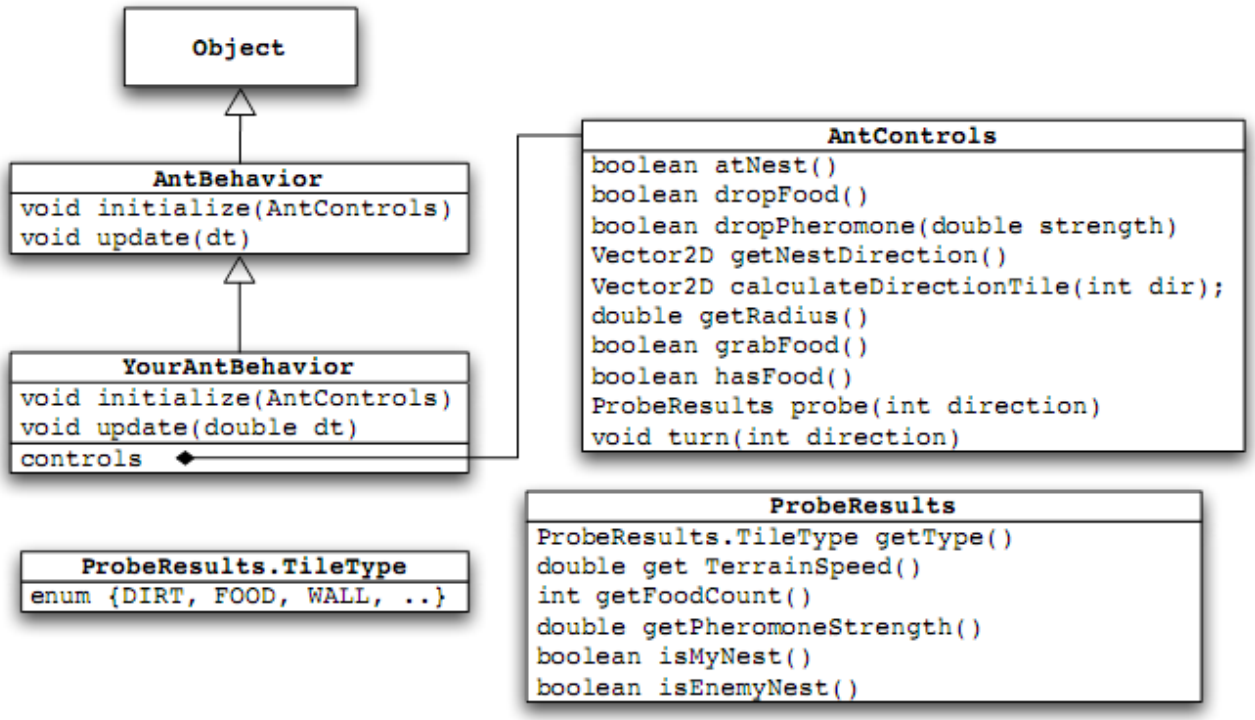

Figure 3.2: *The UML diagram for the ant curriculum. The students extend the AntBehavior class which determines the ant's behavior.*

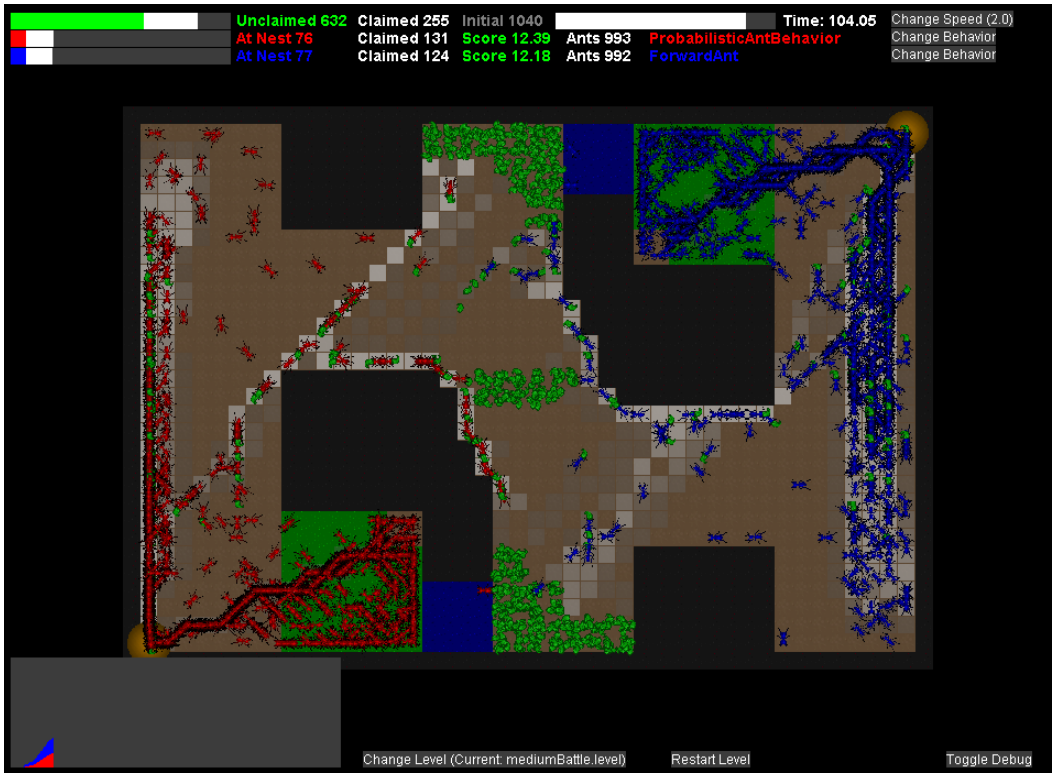

Figure 3.3: *A screenshot of the JIG ants curriculum.*

Question	% A&SA
I enjoy working on projects more when the project produces results that I can display and see (e.g., graphical results).	75
The fact that this project used games as a basis for the programming assignment made the work more interesting to me.	81
The module was interesting to work on.	95

Table 3.1: 2009 Ant Simulation Survey Engagement Questions

3.3 2009 Results

We used the curriculum in two classrooms in the Spring 09 semester, both at Gonzaga and Washington State University Vancouver. At Gonzaga there were 12 male respondents and 2 female, and at WSU Vancouver all 10 respondents were male. The students were given a pre survey prior to working on the assignment, and a post survey after completing the assignment. The full surveys can be seen in the appendix.

3.3.1 Student Engagement

Our first goal was that students find the curriculum engaging and interesting. In the pre survey, 75% of students agreed that "I enjoy working on projects more when the project produces results that I can display and see (e.g., graphical results)." This shows that most students feel graphical results make a project more enjoyable. After taking the assignment, when asked if they agree with the statement "The fact that this project used games as a basis for the programming assignment made the work more interesting to me", 81% of students agreed or strongly agreed with this statement. This high percentage indicates that a majority of students felt the game made the project more interesting. 95% of students also indicated that "The module was interesting to work on."

Question	% A&SA
When decomposing a problem into an object oriented solution, the most intuitive design is the best.	50
Object oriented programming allows modification of an objects behavior independent of the objects model world in which it exists.	56 (91 Post)
Object oriented techniques made it possible for me to implement my solution without worrying about the details of how the project environment/game itself worked.	91

Table 3.2: 2009 Ant Simulation Survey Object Oriented Questions

3.3.2 Object Oriented Techniques

In addition, we wanted to teach object oriented techniques. We found we did not choose the ideal questions for measuring how well the students learned object oriented techniques. For example, we asked the students "When decomposing a problem into an object oriented solution, the most intuitive design is the best.". Only 50% of the students agreed. However, we realized that the best solution is not always "intuitive"; it may take a lot of brainstorming and hard work to come up with a good solution. In another question, we asked with poor wording " Object oriented programming allows modification of an objects behavior independent of the objects model world in which it exists.", however an object can only change its behavior to the extent allowed by the interface. Not to mention the question was poorly worded. 56% of students agreed with this statement while 91% agreed after completing the assignment. Our original intention for these statements was to show that students acquired first hand knowledge that object oriented techniques are powerful and make programming easier, however the wording made the statements less conclusive difficult to comprehend and worse the questions themselves did not do a good job at identifying this knowledge.

On the other hand, students did express some positive attitudes about object oriented programming. When asked if they agree with the statement "Object oriented techniques made it possible for me to implement my solution without worrying about the details of

Question	% A&SA
I feel that I learned a lot by working on the module.	54
The student module was at an appropriate level of difficulty given my knowledge and experience.	58
The student module took a reasonable amount of time to complete.	75
The time required to complete the module was reasonable given how much I learned.	70

Table 3.3: 2009 Ant Simulation Survey Other Questions

how the project environment/game itself worked.", a very high percentage agreed, 91 percent. This shows that the students saw the advantage of object oriented programming when collaborating on a program.

3.3.3 Other Results

Students also responded positively in the open-ended section of the post survey. When asked how this module affected their view of computer science, responses included: "It reminded me how much fun computer science could be.", "[it] [gave] insight into working on [a] small part of [a] program", and "It increased my understanding on [what] teamwork is like in [a] professional environment IE: one team works on one part, another team on other.". However some students responded that it did not affect their view.

A couple students were not as enthusiastic about the assignment however. Two students strongly disagreed with the statement "I feel that I learned a lot by working on the module." The comments from the users showed that one felt "Docs weren't so good so it was hard to tell how to use functions.". The other student responded "Didn't understand how ant worked. Lack of documentation. Methods and had for accessing world information and moving were not easy to work with." Clearly some students felt the documentation was lacking overall.

More students may have found the curriculum difficult as well. In the post survey we

asked the students if "The student module was at an appropriate level of difficulty given my knowledge and experience". Only 58% of students agreed with this statement. We also asked "The time required to complete the module was reasonable given how much I learned.". 70% agreed with this statement. Finally 75% of students felt the "student module took a reasonable amount of time to complete".

The ant behavior class given to solve the JIG ant simulation took an average of 138 lines of code. In comparison, an Objective C implementation of the text based ant simulation took 687 lines of code on average. Because the students do not have to create the simulation environment, their code ends up being much smaller, but they experience a graphical and interactive simulation. This allows students to focus on learning object oriented techniques without worrying about game mechanics, graphics, sounds, and game loops. While learning these topics is important, this curriculum gives the instructor a way to teach object oriented techniques without requiring the students to create a full simulation.

3.4 2010 Improvements

After getting the students comments from the ant surveys and interacting with students, we found that the curriculum was too complex and needed better documentation. Taking the user's comments into consideration, we revised the ant curriculum to make it easier to use in 2010 in two ways.

First, we changed the environment to make it simpler. We changed the maps from being continuous (boundaries could be any polygonal shape) to tile-based. We also simplified the methods the ant used to interact with the world. By making the world tile-based, we reduced the possible ways ants could move which we believe makes debugging easier.

Secondly, we improved the documentation. We ensured every method had complete Java docs, and we created a HTML documentation page that included images, a getting started guide, and tips and explanations about the ant world. Finally, we provided a limited

example ant solution that exhibited basic foraging behavior.

The ant simulator was also improved to allow changing simulation speed, viewing more information about the ants, and more control over world parameters. This made the simulator easier to work with and made testing situations simpler.

As mentioned earlier, we found none of our questions from 2009 adequately measured how well the students had achieved the learning objectives. We were able to modify our 2010 surveys to ask better knowledge questions about object oriented programming both before and after doing the assignment. We also added a direct assessment to our evaluation to measure how well the students did. Finally, we asked a series of focused questions to probe deeper into what might have made the ant curriculum satisfying to students.

3.5 2010 Results and Conclusions

The curriculum was run again in Spring 2010 at Washington State University Vancouver. 16 students took the pre survey(13 male 3 female) and 13 took the post survey (11 male and 2 female). A total of 13 students submitted solutions to the assignment.

3.5.1 Student Engagement

To measure student engagement, we again asked the students if they agreed with the statement "The fact that this project used games as a basis for the programming assignment made the work more interesting to me." 8 students strongly agreed, 3 agreed, 1 was not sure, and one disagreed. With 85% of the students agreeing with the statement, students still found this assignment more interesting because it used games. We did not ask the students if they enjoyed graphical results in the pre survey, but we did ask them if they found the module interesting. Every student either agreed or strongly agreed. In the 2010 survey we also asked the students if the environment made it easier to tell if their solution was correct. Every student taking the post survey agreed that "The game environment made it easy for me to tell if my solution was correct before I turned it in."

Question	2009	2010	Average
The fact that this project used games as a basis for the programming assignment made the work more interesting to me.	81	85	83
The module was interesting to work on.	95	100	97
The game environment made it easy for me to tell if my solution was correct before I turned it in.	N/A	100	N/A

Table 3.4: 2010 Ant Simulation Selected Questions - % Agree or Strongly Agree

3.5.2 Student Satisfaction

We also measured what the students found satisfying about the project. Students were asked to check off the options that they agreed with. Table 3.5 shows some of the most popular selections.

In addition to the 85% agreement with games making the assignment interesting, 85% also found the assignment satisfying because it used games. To reiterate, *students not only thought the game part of the curriculum was interesting, they found it made the assignment more satisfying.* It is interesting to note that one student who marked it as satisfying did not mark it as interesting and another student did the opposite.

77% of students also remarked that the "visuals connected with the module helped me better understand concepts or data." 54% of students thought the module made them think "a lot" 38% even thought the module helped them develop useful professional skills. Clearly many students perceived the assignment as being satisfying and interesting.

3.5.3 Object Oriented Techniques

We also measured the students views on object oriented issues before and after completing the assignment. Students were asked to rate their agreement on a likert scale for various statements about object oriented programming. The results can be seen in Table 3.6

The majority of students seem to indicate they already knew most of the "answers" to the object oriented questions before doing the ant simulation. All students seemed to agree

% of students that selected these attributes as being satisfying	
The module used a game or simulation for a programming project.	85
The visuals connected with the module helped me better understand concepts or data.	77
I could check my answers before turning them in for a grade.	69
The module problem made me think a lot.	54
I feel the module helped me develop useful professional skills.	38
Doing the module gave me practice with key skills or activities.	31
Module activities showed me which concepts I understood well and which I didnt.	31
I could see how the module activities connected to a real world problem.	31
Module activities helped me connect different ideas I had not previously put together.	23
Module activities illustrated concepts we discussed in class.	23

Table 3.5: 2010 Selections for "what about the project made it satisfying for you to work on?"

Statement	%A&SA PRE	%A&SA POST
Object Oriented techniques help to make large project easier to understand.	93	100
Object Oriented techniques allow labor or tasks to be more easily distributed between teams/people.	100	92
Data encapsulation is helpful because it lets programmers know what components/elements they should focus on.	87	85
Object Oriented design allows components (e.g., different sub-classes) to be easily substituted for one another at runtime.	75	100
Providing an interface that documents how methods in a class should behave can make implementation easier.	100	100
Statement	%D&SD PRE	%D&SD POST
Data encapsulation (hiding implementation details) is more trouble than its worth because it keeps programmers from seeing and directly accessing all necessary components.	93	77
Its hard to think of a real world situation in which someone might want to substitute different sub-classes of a particular class at runtime.	62	70

Table 3.6: 2010 WSUV Ants Object Oriented Programming Statements

that Object Oriented programming makes large projects easier to understand, allows labor/-tasks to be more easily distributed, and that interfaces can make implementation easier.

One question that had a larger change in response was “Object Oriented design allows components (e.g., different sub-classes) to be easily substituted for one another at runtime.”. This question had a 25% increase in agreement. It is interesting however, that only 70% disagreed that it is hard to find a “real world” situation that this would matter.

In addition, students viewed data encapsulation as “more trouble than its worth” more after doing the assignment, yet at the same time still agreed that “Data encapsulation is helpful because it lets programmers know what components/elements they should focus on.” This may be due to the fact that the question was asked in the negative form. While some of these questions have conflicting results, the overall response seems to be that the students knew the benefits of object oriented programming. We also wanted to measure the students actual coding habits through direct assessment.

3.5.4 Other Results

Like the previous year, our survey included questions about the module difficulty and if the students felt they learned “a lot”. Students agreed a little more then the previous year that they learned “a lot”, but not much. 77% of students in 2010 thought the module was at an appropriate level of difficulty. Instead of asking if the module took a “reasonable” amount of time to complete, we asked if the student was given adequate time to complete the module. All the students in 2010 agreed with this statement making the average over both years 84%. When factoring in a comparison of time taken versus how much they learned, students agreement level was reduced a little. This may be due to the fact that practicing programming techniques is just as much about experience as it is about head knowledge, thus they may not feel they “learned” a lot even though they gained valuable experience programming in a object oriented language.

Question	2009	2010	Average
I learned a lot by working on the module.	54	61	57
The student module was at an appropriate level of difficulty given my knowledge and experience.	58	77	65
The student module took a reasonable amount of time to complete. (I was given adequate time to complete the module. in Post)	75	100	84
The time required to complete the module was reasonable given how much I learned.	70	75	73

Table 3.7: 2010 Ant Simulation Survey Other Questions - % Agreed or Strongly Agreed

We also asked the students to comment on what particularly interested them. Some of the interesting comments included: "Being able to visually see changes in behavior when making changes to code.", "developing a good strategy for the ants was interesting. It was amusing because their was a slight competition between some of us." and "making the ants get back to their nest because it was satisfying". One student mentioned that he found the challenge of making the ants forage without creating a map and doing path finding made the program even more interesting and challenging.

Students were also asked to comment on what they found particularly difficult. In contrast to the previous year in which students commented on lack of documentation, this year students mostly commented on the difficulty of making a optimal ant. Comments included "The hardest part was mostly just tweaking probabilities trying to find the best general behaviors for the ants given any specific environment.", "it was difficult to get ants out of a cavernous area when they had to go the wrong way to get to their nest because was surrounded by so many walls." and "Debugging was difficult. It was hard to tell why and if an ant was not behaving correctly.". Although one student found debugging difficult, the overall response was greatly improved. *No students mentioned problems with getting the environment running or documentation.*

3.5.5 Direct Assessment

To do our direct assessment, we looked at each submitted assignment and graded it as "Excellent", "Proficient", or "Apprentice" in two areas.

First, were the students able to create a class that implemented all the methods, used well structured code that was readable, and used the world object given to interact with the world. Solutions that did not compile or did not differ more than a couple lines of code from the given example ant were graded as apprentice. Solutions that implemented the methods and differed from the example ant were graded as proficient. Excellent solutions were very readable and maintainable (comments, constants, etc.) had well structured code, and had a larger degree of complexity.

Second, we look at if the students use debugging tools and feedback from the simulation to get their ant solution to adequately forage for food. It should be noted that creating a good foraging algorithm is not the point of the ant exercise. Rather, we want the students to learn object oriented techniques in the process of creating a good foraging algorithm. To create a good algorithm, students must understand all the interface methods well, they must study the objects, create a semi-complex ant class, and evaluate what their code is doing in the simulator and improve it. All of these things help them learn programming techniques. Their foraging strength is just an indicator of this fact.

Six maps were given to the student as well as the example ant. We created a seventh map and compared the student's scores on each map with the example ant. We ran the solutions on all seven maps and calculated their average score. We ranked a solution as apprentice if it did not beat the example ant's score, and proficient if it beat the example ant. Finally a excellent solution also beat a good ant solution we had developed.

Table 3.8 shows the results of the direct assessment. 76 percent of students achieved the excellent level for Object Oriented Techniques. The student's source code was mostly well

Outcome	Excellent	Proficient	Apprentice
Object Oriented Techniques	76%	15%	7%
Foraging Behavior	38%	61%	7%

Table 3.8: 2010 Ant Simulation Direct Assessment

documented and had extra methods added for structure. In addition many students added constants and named their fields with readable names.

Overall the student's ants foraged very well as well. 61 percent beat the example ant, most by a wide margin. In addition, 38 percent even beat the advanced solution we had made. The top ranked solution's source code had over 800 lines. Clearly some of the students spent a significant amount of time optimizing their solutions.

The combination of the surveys and direct assessment show that the students both understand the basics of object oriented programming and found the assignment to be satisfying and engaging.

CHAPTER FOUR

THE LIST PUZZLE GAME

4.1 List Puzzle Background

The goal of the list puzzle game is to teach students an how to program an abstract list data type while making the results interesting. The puzzle game was designed so that the player's actions correspond directly to operations supported by the abstract data type the students implement. Thus, only a correct list implementation results in a fully functioning game.

Figure 4.1 shows a screenshot of level 3 of the game. The player's goal is to arrange different shapes in separate columns so that the numbers attached to each shape are in ascending order. The player moves shapes around by dragging them with a mouse. Dragging a shape splits a column into two parts: a head which remains stationary, and a movable tail containing one or more shapes that can then be attached to the front or rear of another column. In addition, the player may use a power-up once per level that enables her to extract a single element from a column and insert it into beginning or end of any column.

Our abstract list data type contains six methods described below that together provide the functionality for the game:

removeRemaining(predecessor): splits a list into head and tail segments.

removeNext(predecessor): provides the functionality for the power-up move.

insertAfter(predecessor, listToInsert): allows a floating list segment to be spliced into a new column.

findPredecessor(searchfrom, selected): finds the list element that the user has selected with the mouse.

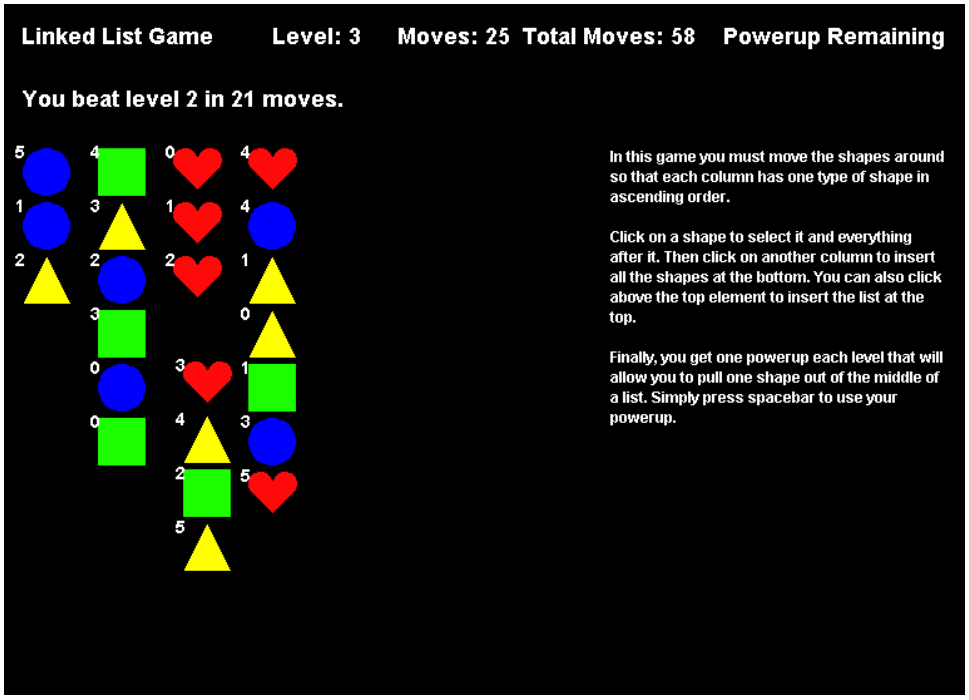

Figure 4.1: *A screenshot of the list puzzle game.*

getHead(): used with insertAfter to splice at the front of a list.

getLast(): used with insertAfter to splice at the end of a list.

Aside from the feedback students receive by playing the game and checking which actions are functional, the game also provides rigorous error checking. By testing the validity of lists after each of the player's actions, the game ensures that the lists always remain in a consistent state. If they do not (e.g., because a node gets orphaned or appears in two lists simultaneously) diagnostic information is displayed on the screen to help a student find the source of their error. Just like the ant curriculum, this game is plug-n-play. The student compiles their solution in the correct folder and the game loads it automatically.

4.2 Results

To perform our direct assessment, we selected learning objectives and scored the students work based on rubrics pertaining to these outcomes. We then categorized achievement on a scale borrowed from the Miami University Computer Science program. Namely this scale included: “Excellent”, “Proficient”, “Apprentice”, and “Novice” categories. Ideally, most students should be rated “Excellent” or “Proficient”.

At WSUV, there were two learning objectives. First, did the student “Select and implement appropriate fundamental data structures to solve problems.” If students correctly implemented all the functionality, they were put in the Excellent or Proficient category. Secondly, did the students demonstrate good coding style and readability. If they did, they were put in the upper categories, but if the code style indicated a misunderstanding or lacked readability the student was marked as Apprentice or Novice.

At Miami, students were to “Implement basic control structures, nested control structures, well-specified classes, single and multi-dimensional arrays”. Excellent meant that the student was able to implement all the required functionality. Proficient meant that most of the assignment worked but there may have been a problem with some aspects of list implementation. Apprentice meant that some things worked OK but the game was not playable and Novice meant that nothing really worked. Finally, students were to use the ArrayList API correctly. If they did not, they were marked as Apprentice or Novice.

The assessment results for WSUV and Miami University are presented in Table 4.1. Our assessment showed that the majority of the students achieved the desired learning outcomes. This shows that the assignment served its purpose. Futhermore, we evaluated student satisfaction similar to our ant curriculum.

The 13-question survey consisted of 10 questions ranked on a Likert Scale (1: Strongly Disagree to 5: Strongly Agree), one question where students selected reasons why they

	Outcome	Excellent/Proficient	Apprentice/Novice
WSUV	Select and implement appropriate fundamental data structures to solve problems.	88% Correctly implemented all list functionality ensuring that the game worked.	12% Critical list functionality was not implemented correctly. The game does not work.
		77% Demonstrated good coding style and readability.	23% Code style indicated a misunderstanding or lacked readability.
Miami	Implement basic control structures, nested control structures, well-specified classes, single and multi-dimensional arrays	87% Correctly implemented all list functionality ensuring that the game worked.	13% Critical list functionality was not implemented correctly. The game does not work.
	Use class libraries to assist in problem solving.	95% Used ArrayList correctly to implement list.	5% Did not use Java ArrayList API correctly.

Table 4.1: List Puzzle Game Direct Assessment

Question	Miami %A&SA			WSUV
	Total	Men	Women	%A&SA
The project was interesting to work on.	96%	94%	100%	88%
The project contributed to my overall understanding of the material in the course.	92%	89%	100%	88%
This project was more interesting to work on because it involved games.	88%	83%	100%	88%
I had a positive learning experience in this course.	79%	78%	83%	88%

Table 4.2: Student Perceptions of the Puzzle Game Project

thought the project was satisfying to work on, and two open-ended questions. The survey was issued to all students taking the relevant courses at both universities; there were 24 respondents (18 male; 6 female) at Miami University and 8 respondents (all male) at WSU Vancouver.

The majority of students from both universities were enthusiastic about the project and felt that it contributed to their understanding of the material in the course. Similarly to our Ants results, most students found this project to be more interesting because it involved games. Table 4.2 shows percentages of students who replied with "Agree" or "Strongly Agree" to 4 select questions. There is a potential concern about how game-based curriculum may impact women in computing [12]. There were 6 women respondents at Miami University and Table 4.2 shows total results for Miami students as well as separate results for men and women in CS2. The results indicate that, at least in the case of this assignment, this cohort of women students was even more enthusiastic than the men about the project.

Question 11 of the survey "What about the project made it satisfying for you to work on? (Check all that apply)" presented several options for students to choose from just like the 2010 ant curriculum survey. Interestingly, students on both campuses agreed on the top four reasons as to why this project was satisfying; these are summarized in Table 4.3. Another highly ranked response by WSUV students was "The visuals connected with

Answer	Miami %A&SA	WSU %A&SA
The project used a game or simulation for a programming project.	74%	100%
I could see how the project activities connected to a real world problem.	65%	63%
I could check my answers before turning them in for a grade.	52%	75%
Project activities illustrated concepts we discussed in class.	52%	75%

Table 4.3: Most Popular Selections for "what about the project made it satisfying for you to work on?" (Question 11)

the project helped me better understand concepts or data" (75%). A significantly smaller proportion of students at Miami University selected this response (30%), perhaps because their assignment did not use linked lists directly.

Together, the results from our survey indicate that students felt that the puzzle game assignment enhanced their learning experience and that the games-based project was more interesting than other course projects. These results are supported by direct measurements of the student's performance on these assignments.

CHAPTER FIVE

IMPROVING THE JIG API

5.1 Previous Work

Improving the JIG engine is an important goal of the overall JIG project. In this section we discuss previous work done to improve APIs including good API design practices and predicting problem spots in APIs. We then discuss a tool we created to improve the JIG API and report on its results.

5.1.1 Good API Design

Clarke, a member of the Visual Studio usability group, and his team discuss API usability in their May 2004 article [7]. They suggest that understanding the expectations of the users of a developer's API is the most important part of API design. They encourage API designers to think about how they want code that uses their API to look before they even start creating the API. This sample code should meet the expectations of the API's future users. Does it have a reasonable number of lines of code? Is it readable to someone unfamiliar with the API? These are a couple examples of the 12 cognitive dimensions (attributes about APIs) the paper presents. If these factors are considered before writing the API, the API is more likely to be developed based on the users needs and not the implementation details. Additionally, after the API is completed it can be compared with the customer's expectations by ranking these cognitive dimensions and graphing them.

While we were not able to do a full study on the API using this method, we can still apply a conceptual evaluation after the fact. We will consider users first using the JIG engine in a game design class. The target users are on average Junior level computer science students with some experience with Java. We select a subset of the cognitive dimensions, describe them, and analyze likely student expectations vs the actual API below.

Learning Style What are the learning requirements of the API? The students want a API that uses simple coding techniques in Java since they only have limited knowledge of the language. The JIG engine requires a understanding of Java, including some use of generic types, but most uses do not require knowledge about threads, synchronization, or reflection. The use of examples, will help students understand generics.

Work-step Unit This describes how much coding must be done to get one goal in the programming project done. Students want this to be low so they don't feel discouraged. To create a initial game like "Pong" takes 60 lines of source code in JIG. An example game is usually provided and work-steps after that are not as big. In comparison, an example Swing Pong demo given on-line took 133 lines of source code [8].

Progressive Evaluation Can the code be iterated on easily to allow a transition toward the final solution. The students definitely want this. The ant curriculum easily provides this, however in game design this can sometimes be harder. However usually once the initial game starts to come together, progress can be seen easier such as adding sound, graphics, and other details. In the case of the ant game, the programmer can slowly add complexity to the ant solution, and observe the results. The student might start with making sure the ants get to the food, then work on getting them to come back to the nest, and finally add traversing water and other complex steps.

5.1.2 Common API Problems

In Christopher Scaffidi's paper, he talks about the problems many users face when dealing with APIs. [18] In the following section I list the problems he gives and the ways he says users and designers try to solve them. When designing APIs it would be wise for us to keep these problems in mind.

Documentation Users often get frustrated with documentation that is lacking. At the same

time there is always pressure to get APIs "out the door" and thus documentation is often lacking. Users often consult various documents that overlap to get a better understanding of the API. It is important that designers not only provide good documentation, but also examples and a way for the community to discuss problems and post code.

Coupling between API calls Sometimes APIs can have undesired side-effects between two API calls. The author gives the example of databases committing all changes before they make a table structure change. Users should analyze which API methods they need and try to only use coupled APIs when they will not cause undesired side-effects. API designers should try to reduce coupling to only what is necessary and should document it fully to assist the user.

Not enough abstraction Abstraction is an important part of API design because it helps ensure the API will be flexible in the future. Specifically the author mentions how an API should be able to do the general task, not just the specific. For example a cryptography API should be able to decrypt a stream with a key, not just do a specific decryption task like follow a protocol. Users often deal with this by creating their own code to make the correct abstraction. Designers can try to avoid this problem by planning ahead for what users will need. If the designers are worried about exposing too many methods to the average user, they can use a Facade pattern to provide a simpler form for the basic user. The Facade pattern is a pattern that reduces the complexity of a class by creating another class (the facade) that has a simplified interface.

The JIG API does relatively well in some of these areas, in others it still needs improvement. For documentation, the entire JIG engine has documentation generated by the javadoc tool. In addition, the website contains example code for a couple games as well

as common tasks. However, as we saw in our first iteration of the ants curriculum and the upcoming evaluation of the JIG API, documentation still needs work. We find in our results in section 5.3 that 15.3% of the methods in our API needed improvement to their documentation.

The API has no major coupling issues. This has been avoided to make the engine easier to use for beginning programmers. For example a group of objects in the JIG engine are often grouped in what is called a "layer". These layers are used to draw and manage the objects. If an object were to be removed from a layer while the layer was being used by another thread, an error would occur. To remove this coupling issue, a remove method is not in the layer interface. Thus students are able to avoid this coupling issue. If the student needs to remove objects later on in their studies, the student may still add the remove method to their own implementation of the layer interface after they understand the coupling issues.

There are 28 abstract classes or interfaces in the JIG API giving it a high degree of abstraction. Students can use the simple static screen game with the default time managers and rendering options, or they can go into the details and manage the game clock, transform the rendering coordinates, or create their own game loop if needed. This gives the students the power to do what they want, but also the simpler versions if they do not need the extra features.

5.1.3 Predicting Changes

Many papers have presented results on predicting the maintainability of code based on metrics. Nikolaos Tsantalis et al.'s paper on Predicting the Probability of Change in Object-Oriented Systems is very insightful [22]. In the paper, the authors detail how they can use metrics about Java code to predict the probability a class will need to change based on how it relates to other classes in the code. This helps to predict which classes are more stable

and which classes are more likely to change causing users to fix code. To measure the probability, the authors uses three axis of change: inheritance — interfaces or subclasses are more likely to need change; constructors and references — if a class uses another class, it is more likely to need change; Dependency — if a class is a inner class or has a package, then it is dependent on that larger structure. Using these three axis, the authors predict the probability of change on classes accurately when compared to the actual changes made by source history. In future work, this method should be compared with the method we describe in the next section.

5.2 Design

5.2.1 Method Fault Classification

Keeping good API design in mind, we set out to improve the JIG project. Using experience with students and the API, we came up with a set of reasons that a method might need improvement. The following are the categories we settled on.

Visibility Methods in this category have the wrong visibility. This means they are either public when they could be internal, or they are internal even though they really should be accessible in the API.

Functionality When developing curriculum frameworks often times features can be created that may never be used. Methods in this category do not add any necessary features, thus they only complicate the system.

Documentation Methods should have full documentation with all variables explained including proper units and return values. Methods with incomplete or no documentation are in this category.

Redundancy Methods in this category have relatively low lines of code. They are usually

specialized or simplified versions of other methods for convenience. While this is sometimes useful, it can also just complicate the system depending on the situation.

After creating these categories, the next step in improving the API was identifying the methods and fixing them. With small libraries, a full inspection of the code base is simple, however with massive libraries, this quickly becomes a burden. It would be ideal to have a way to predict which methods will need improvement.

5.2.2 Predicting Problems

We hypothesize that methods referenced less in typical client code are more likely to need improvement. This seems like a reasonable hypothesis, since well used methods are more likely to be well tested through daily use, and methods with bad documentation or other problems are more likely to go unused.

To test this hypothesis every method in the JIG library was examined to see if it has any of the problems listed earlier. We manually identified every method and recorded the results.

We created a Eclipse plug-in to list all methods in a project and count all references to each method. This plug-in is made freely available to the public at my website. [5]. The plug-in uses Eclipse's built in Java Development Tools to search all open projects for references to each method. It then sorts the list of all methods and displays them in a development window in the IDE. A screenshot of the window is shown in Figure 5.1

We collected student games, our demos and curriculum modules all of which use the JIG library. We analyzed how many times every method in the JIG library was called using our eclipse plug-in.

Our heuristic allows us to visit API methods based on how often they've been used. If our hypothesis is correct, then by visiting seldom used methods first, we will examine more methods that require fixing sooner. In this way, we hope to be able to put a limited amount

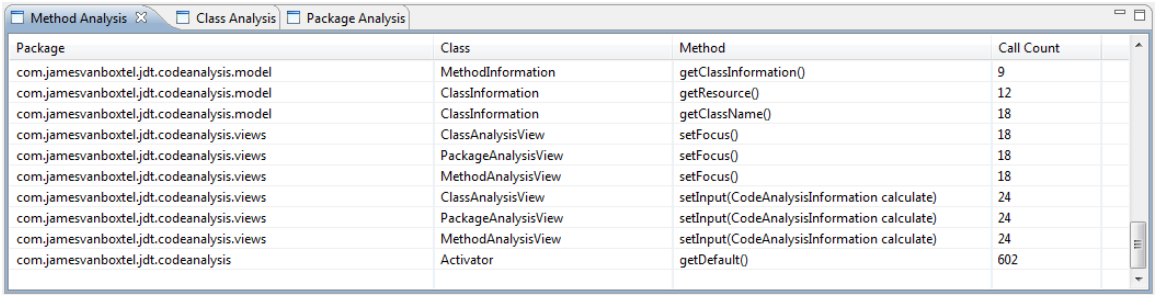

Figure 5.1: A screenshot of the reference count plugin in Eclipse

No Problems	Visibility	Functionality	Documentation	Redundancy
615	20	7	121	29

Table 5.1: Classification of Methods

of resources, say 5 hours of time, to its best possible use. To test the value of our heuristic, we compare how many errors are found using the heuristic order vs random orderings.

5.3 Evaluation

5.3.1 Method Categorization

After categorizing all the JIG methods we found 177 needed improvement out of 790 methods. That is 22.4% of all the methods needed improvement. The biggest problem was documentation, followed by complexity and then visibility. Functionality problems were rare. Note methods could have multiple problems, thus the total in the table does not equal the total methods.

5.3.2 Method Improvement Heuristic

The graph in figure 5.2 shows the percentage of problem methods found vs the percentage of methods evaluated. An ideal heuristic would increase very quickly at the beginning indicating it found all the problem methods first. The least used methods heuristic is compared with the optimal performance and random ordering. To obtain the random plots, methods

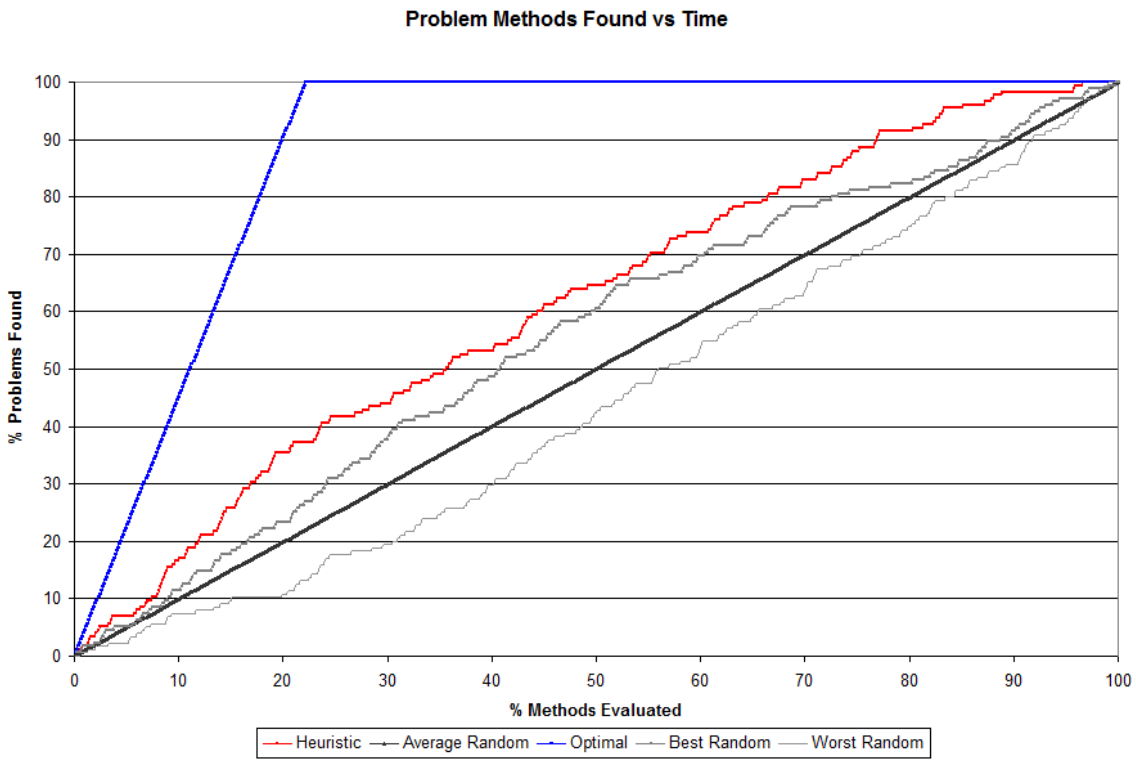

Figure 5.2: Graph showing the percentage of problem methods found over the total percentage of methods tested.

were randomly ordered 500 times. The best and worst results were graphed, along with the average value of all 500 runs. The random plot shows that the average random selection has a linear rate of finding problem methods.

The heuristic was most effective in the first 20% of the methods where it found 35.42% of the problem methods compared to the optimum performance which would find 90.3% of the problem methods. As the search continued, diminishing improvements over random orderings were seen.

To compare the different methods we calculate the area under each curve. This value can be treated as a relative "Score" of each ordering method and the results are presented in Table 5.2. We show the distribution of the scores from the random orderings and compare

Worst Random	Best Random	Heuristic	Optimal
44.16	55.71	61.34	88.99

Table 5.2: Relative 'Score' of Different Ordering Methods

them with the heuristic and optimal performance in Figure 5.3. The random orderings are grouped into 10 buckets along with the number of orderings within each bucket. The average random score is 50.07 and the standard deviation is 1.96. This clearly shows that our heuristic is outside of random chance as we are more than 5 standard deviations away.

5.3.3 Shortcomings of the Experimental Design

While this experiment does have some interesting results, it also has some shortcomings. These shortcomings are mainly a result of lack of time and man-power and could be resolved in future work. There are two main shortcomings.

First, the scope of the experiment is quite narrow. Ideally this experiment would be repeated with a wide variety of libraries. Unfortunately this would take a significant investment of time, but with collaboration this could be reduced.

Second, there is an issue with the subjectivity of classifying methods. Indeed, one person may feel a method has good documentation while another may find it lacking. Or, one person may find a method unnecessary or redundant, while another may feel it provides enough convenience to keep it in the library. We feel that having more users involved in the process can help resolve this issue.

However, we feel that these results do indicate that low method usage is a predictor of problem methods despite these shortcomings. Our results show that this ordering outperforms random ordering by a significant margin. Our heuristic is also simple, which makes this heuristic easy to apply since many IDEs allow you to look up the number of references to a method.

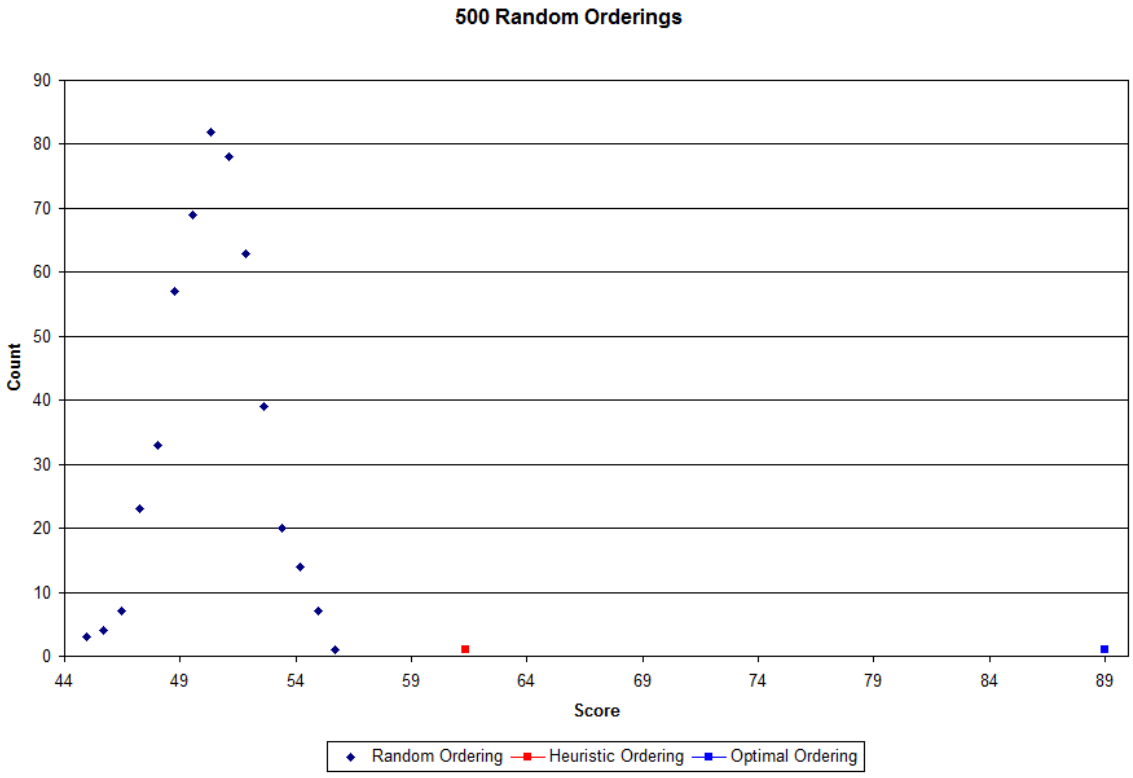

Figure 5.3: Graph showing the distribution of random orderings. The heuristic presented is outside of the standard distribution.

CHAPTER SIX

CONCLUSIONS

6.1 Interactive Curriculum

The results of our research in interactive curriculum indicate it is not only engaging to students, but assignments can also serve their purpose in teaching learning objectives.

If we combine all results on "The fact that this project used games as a basis for the programming assignment made the work more interesting to me" and "This project was more interesting to work on because it involved games." we find 86.1 percent of students agreed. Clearly students felt the interactive nature of these games was engaging.

Direct assessment also showed students understood what was being taught. In the ant curriculum, 92.3% of students achieved a Proficient or better level in both learning objectives. The list puzzle game had a weighted average of 88.8% for all learning objectives at both universities. With almost ninety percent of students achieving high levels of success, it seems clear that the interactive curriculum was effective at teaching the concepts.

More research should be done in evaluating what makes curriculum engaging. For example, did the students not find "I could see how the module activities connected to a real world problem." satisfying because the problem wasn't "real world" or was it because that situation is not satisfying? There are many more questions to be answered about satisfying curriculum.

Student learning should be investigated more in future research as well. We did not get a chance to compare learning objectives directly with non interactive curriculum. For example, a experiment could be run where students create a blackjack computer player in a text based environment or graphical environment. It could then be tested which program the students learn from better. This would allow a more direct comparison of learning

objectives across curriculum types.

6.2 API Improvement

Our method for predicting problem methods in an API also shows promise. It found 1.7 times as many problem methods than randomly examining 20% of the methods in an API. Clearly this is an improvement if no prior knowledge about problem methods is known. However, in the future other attributes and prior knowledge could be factored in to improve our heuristic. For example, number of lines of code in each method may also help in predicting error prone methods. Methods that have low lines of code may be able to be re-factored, and methods with high lines of code may be too complex and have errors.

6.3 Final Thoughts

Interactive curriculum has become a truly powerful teaching method over the years. It has the power to engage, capture the student's imagination, and can even bring out students competitive nature. There is a chinese proverb that says "Tell me and I forget. Show me and I remember. Involve me and I understand." That is exactly what interactive curriculum is about, and the more teachers embrace it, the more they will understand it's power to teach.

APPENDIX

APPENDIX ONE

2009 ANT CURRICULUM SURVEY

Table A.1: To what extent do you agree or disagree with the statements below?

Pre Survey					
SD = Strongly Disagree D = Disagree Somewhat N = Not sure A = Agree Somewhat SA = Strongly Agree					
	SD	D	N	A	SA
I enjoy working on projects more when the project produces results that I can display and see (e.g., graphical results).	1	1	4	6	12
I enjoy assignments more when I can see how they tie into a real world problem.	1	0	0	11	10
Computer science/computing requires a lot of creative thought.	1	0	1	11	10
Computer models of biological processes can be used to gain insight into real world behavior.	0	1	2	9	10
When decomposing a problem into an object oriented solution, the most intuitive design is the best.	2	4	6	9	3
Object oriented programming allows modification of an objects behavior independent of the objects model world in which it exists.	0	4	6	9	4
SD = Strongly Disagree D = Disagree Somewhat N = Not sure A = Agree Somewhat SA = Strongly Agree					

Table A.2: This section asks for feedback about the module you did. Please indicate what is true for you.

Post Survey					
SD = Strongly Disagree D = Disagree Somewhat N = Not sure A = Agree Somewhat SA = Strongly Agree					
	SD	D	N	A	SA
The module was interesting to work on.	0	0	1	15	8
I feel that I learned a lot by working on the module.	1	1	9	10	3
The student module was at an appropriate level of difficulty given my knowledge and experience.	2	4	4	8	6
The student module took a reasonable amount of time to complete.	0	0	6	14	4
The time required to complete the module was reasonable given how much I learned.	0	1	6	10	7
The fact that this project used games as a basis for the programming assignment made the work more interesting to me.	0	0	4	9	9
Object oriented techniques made it possible for me to implement my solution without worrying about the details of how the project environment/game itself worked.	1	1	0	9	12
Object oriented programming allows modification of an objects behavior independent of the objects model world in which it exists.	0	1	1	11	10

APPENDIX TWO

2010 ANT CURRICULUM SURVEY

Table B.1: This section asks for feedback about the module you did. Please indicate what is true for you.

Post Survey					
SD = Strongly Disagree D = Disagree Somewhat N = Not sure A = Agree Somewhat SA = Strongly Agree					
	SD	D	N	A	SA
This module was interesting to work on.	0	0	0	5	8
I learned a lot by working on the module.	1	1	3	3	5
The module was at an appropriate level of difficulty given my skills, knowledge and experience.	1	1	1	6	4
I was given adequate time to complete the module.	0	0	0	5	8
The time required to complete the module was reasonable given how much I learned.	0	0	3	6	4
Before taking this course, I had done something that required thinking skills or computing skills similar to those I used in the module.	0	0	1	5	7
Before taking this course, I had done something that required team skills or communication skills similar to those I used in the module.	1	2	1	3	6

The fact that this project used games as a basis for the programming assignment made the work more interesting to me.	0	1	1	3	8
The game environment made it easy for me to tell if my solution was correct before I turned it in.	0	0	0	7	6

Table B.2: What about the project made it satisfying for you to work on? (13 Responses)

The module used a game or simulation for a programming project.	11
The visuals connected with the module helped me better understand concepts or data.	10
I could check my answers before turning them in for a grade.	9
The module problem made me think a lot.	7
I feel the module helped me develop useful professional skills.	5
Doing the module gave me practice with key skills or activities.	4
Module activities showed me which concepts I understood well and which I didnt.	4
I could see how the module activities connected to a real world problem.	4

Module activities helped me connect different ideas I had not previously put together.	3
Module activities illustrated concepts we discussed in class.	3

Table B.3: For each statement about Object Oriented programming below, put an X in the column that corresponds to your level agreement with the statement.

Pre Survey					
SD = Strongly Disagree D = Disagree Somewhat N = Not sure A = Agree Somewhat SA = Strongly Agree					
	SD	D	N	A	SA
Object Oriented techniques help to make large project easier to understand.	0	0	1	8	7
Object Oriented techniques allow labor or tasks to be more easily distributed between teams/people.	0	0	0	8	8
Data encapsulation (hiding implementation details) is more trouble than its worth because it keeps programmers from seeing and directly accessing all necessary components.	4	11	1	0	0
Data encapsulation is helpful because it lets programmers know what components/elements they should focus on.	0	0	2	11	3
Object Oriented design allows components (e.g., different sub-classes) to be easily substituted for one another at run-time.	0	1	3	6	6

Its hard to think of a real world situation in which someone might want to substitute different sub-classes of a particular class at runtime.	3	7	4	1	1
Providing an interface that documents how methods in a class should behave can make implementation easier.	0	0	0	6	10
Post Survey					
Object Oriented techniques help to make large project easier to understand.	0	0	0	5	8
Object Oriented techniques allow labor or tasks to be more easily distributed between teams/people.	0	0	1	4	8
Data encapsulation (hiding implementation details) is more trouble than its worth because it keeps programmers from seeing and directly accessing all necessary components.	5	5	0	0	3
Data encapsulation is helpful because it lets programmers know what components/elements they should focus on.	0	0	2	5	6
Object Oriented design allows components (e.g., different sub-classes) to be easily substituted for one another at run-time.	0	0	0	5	8
Its hard to think of a real world situation in which someone might want to substitute different sub-classes of a particular class at runtime.	5	4	2	2	0
Providing an interface that documents how methods in a class should behave can make implementation easier.	0	0	0	5	8

APPENDIX THREE

2010 LIST PUZZLE GAME SURVEY

Table C.1: To what extent do you agree or disagree with the statements below?

SD = Strongly Disagree D = Disagree Somewhat N = Not sure A = Agree Somewhat SA = Strongly Agree					
	SD	D	N	A	SA
Time allowed for completion of the project was sufficient	1	1	1	8	20
The project was interesting to work on.	1	0	1	11	18
The project contributed to my overall understanding of the material in the course.	1	1	1	16	12
The project took a reasonable amount of time to complete.	1	1	1	13	15
The project was at an appropriate level of difficulty given my knowledge of computer science and programming.	1	3	4	9	13
This project was more interesting to work on because it involved games.	1	0	2	9	18
I liked working on this project more than the average Computer Science assignment.	1	2	6	10	11
I had a positive learning experience in this course.	1	1	3	12	13
I learned a lot by working on this project.	2	0	8	15	5
The time required to complete this project was reasonable given how much I learned.	1	1	6	14	8

Table C.2: What about the project made it satisfying for you to work on?

The project used a game or simulation for a programming project.	25
I could see how the project activities connected to a real world problem.	20
Project activities illustrated concepts we discussed in class.	18
I could check my answers before turning them in for a grade.	18
The project problem made me think a lot.	16
Doing the project gave me practice with key skills or activities.	15
The visuals connected with the project helped me better understand concepts or data.	13
Project activities showed me which concepts I understood well and which I didnt.	13
Project activities helped me connect different ideas I had not previously put together.	9
I feel the project helped me develop useful professional skills.	8

BIBLIOGRAPHY

[1] Tiffany Barnes, Heather Richter, Eve Powell, Amanda Chaffin, and Alex Godwin. Game2learn: building cs1 learning games for retention. *SIGCSE Bull.*, 39(3):121–125, 2007.

[2] Theresa Beaubouef and John Mason. Why the high attrition rate for computer science students: some thoughts and observations. *SIGCSE Bull.*, 37(2):103–106, 2005.

[3] Katrin Becker. Teaching with games: the minesweeper and asteroids experience. *J. Comput. Small Coll.*, 17(2):23–33, 2001.

[4] The College Board. Computer science a & ab: Introduction of java in 2003-2004. `http://apcentral.collegeboard.com/apc/members/courses/teachers_corner/4614.html`.

[5] James Van Boxtel. Reference count eclipse plugin. `http://jamesvanboxtel.com/projects/reference-count-eclipse-plugin/`, March 2010.

[6] Kim B. Bruce. Controversy on how to teach cs 1: a discussion on the sigcse-members mailing list. *SIGCSE Bull.*, 37(2):111–117, 2005.

[7] S Clarke. Measuring api usability. *Dr. Dobb's Journal Windows/.NET Supplement*, pages S6–S9, May 2004.

[8] CodeCall.net. A simple pong game. `http://forum.codecall.net/java-tutorials/14191-game-simple-pong.html/`, 2010.

[9] PAUL GESTWICKI and FU-SHING SUN. Teaching design patterns through computer game development. *Ball State University*, 2008.

[10] Ray Giguette. Pre-games: games designed to introduce cs1 and cs2 programming assignments. *SIGCSE Bull.*, 35(1):288–292, 2003.

[11] Patricia Haden. The incredible rainbow spitting chicken: teaching traditional programming skills through games programming. In *ACE '06: Proceedings of the 8th Austalian conference on Computing education*, pages 81–89, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[12] Susan Haller, Brian Ladd, Scott Leutenegger, John Nordlinger, Jody Paul, Henry Walker, and Carol Zander. Games: good/evil. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 219–220, New York, NY, USA, 2008. ACM.

[13] Poul Henriksen and Michael Kölling. greenfoot: combining object visualisation with interaction. In *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 73–82, New York, NY, USA, 2004. ACM.

[14] Timothy Huang. Strategy game programming projects. *J. Comput. Small Coll.*, 16(4):205–213, 2001.

[15] Ricardo Jiménez-Peris, Sami Khuri, and Marta Pati no-Martínez. Adding breadth to cs1 and cs2 courses through visual and interactive programming projects. In *SIGCSE '99: The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 252–256, New York, NY, USA, 1999. ACM.

[16] Ian Parberry, Timothy Roden, and Max B. Kazemzadeh. Experience with an industry-driven capstone course on game programming: extended abstract. *SIGCSE Bull.*, 37(1):91–95, 2005.

[17] N. Parlante. Nifty assignments. `http://nifty.stanford.edu/`, 2010.

[18] Christopher Scaffidi. Why are apis difficult to learn and use? *Crossroads*, 12(4):4–4, 2006.

[19] Kurt Squire, Levi Giovanetto, Ben Devane, and Shree Durga. From users to designers: Building a self-organizing game-based learning environment. *TechTrends*, 2005.

[20] Kelvin Sung, Michael Panitz, Scott Wallace, Ruth Anderson, and John Nordlinger. Game-themed programming assignments: the faculty perspective. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 300–304, New York, NY, USA, 2008. ACM.

[21] Elizabeth Sweedyk, Marianne deLaet, Michael C. Slattery, and James Kuffner. Computer games and cs education: why and how. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 256–257, New York, NY, USA, 2005. ACM.

[22] Nikolaos Tsantalis, Alexander Chatzigeorgiou, and George Stephanides. Predicting the probability of change in object-oriented systems. *IEEE Trans. Softw. Eng.*, 31(7):601–614, 2005.

[23] Scott A. Wallace, Robert Bryant, and Genevieve Orr. The northwest distributed computer science department. *J. Comput. Small Coll.*, 25(1):143–148, 2009.

[24] Scott A. Wallace and Andrew Nierman. Addressing the need for a java based game curriculum. *Journal of Computing Sciences in Colleges*, 2006.

[25] Scott A. Wallace, Ingrid Russell, and Zdravko Markov. Integrating games and machine learning in the undergraduate computer science classroom. In *GDCSE '08: Proceedings of the 3rd international conference on Game development in computer science education*, pages 56–60, New York, NY, USA, 2008. ACM.