Project Number. GFP0810

Teaching Programming Technique

A Interactive Qualifying Project Report:

submitted to the faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by:

Isaac Edwards

Andrew Mendelbaum

Christopher Trufan

Date: May 05, 2009

Approved:

Professor Gary F. Pollice, Major Advisor

1. Teaching
2. Programming
3. Technique

Abstract

This project was created with the goal of increasing basic knowledge of programming and increasing interest in programming in high school and middle school aged children. To achieve this goal, we designed a program to teach the user some of the basic concepts of programming and object oriented thinking and created a study to measure the effectiveness of the program. We achieved an average increase of 20% in basic knowledge of concepts we taught as well as a mild increase in interest.

This report outlines the goals of our Interactive Qualifying Project *Teaching Programming Technique*. It covers existing program based teaching aids, the strengths and weaknesses of the existing programs, and the features we were looking to develop in our own program. It details the goals and purposes of our initial project and how we adapted our goals based on our research. It explains how we went about designing a program to teach object oriented programming without the steep learning curve many existing methodologies have, and the processes by which we tested the effectiveness of the program. It details our testing procedure, and concludes with a critical evaluation of the results, whereby we elaborate on ways our program could potentially be improved or used as a platform for future development.

Acknowledgements

This project would not have been possible without the help of the following people:

- Gary Pollice, advisor
- Jeanine Skorinko, who provided access to a large group of test subjects and the means to communicate and schedule with them
- Billy Hnath and the ACM, allowed us to use their room for our study
- Sarah Goodwin, who helped make sure that the script was understandable
- Sarah Tung, who helped make sure that the script was understandable

Table of Contents

1. Introduction

This project was designed as an interactive program to introduce basic object oriented programming concepts as well as spark an interest in programming amongst middle school and high school aged children. There is a written portion on the right panel that introduces the concepts and explains them via written text as well as providing a few examples in actual code. On the left panel there is a playable RISK game. Rules and playing instructions are provided, and the game is simplified so that users can focus on learning object oriented programming rather than the rules of a game. The user is also presented with samples of code that they can modify and witness the results of their modifications in the playable game panel. With the code sections carefully chosen, the results are very noticeable and help to visually connect the more abstract code to the objects of the RISK game. This helps the user to think in terms of objects, a key requirement to learning object oriented development. After the user gets used to editing existing code they are asked to start writing their own. The RISK game is replaced by a game of Blackjack. Aided by prompts, the user must write sections of the code for the Blackjack game. This allows the user to feel as though they are writing the actual program and not just doing minor changes, as well as testing the applicable knowledge they have gained by running through our program. Throughout all this the accompanying text helps the user so that they do not get lost or frustrated, even if they do not have someone (such as a teacher) to help guide them. At the end of the RISK game and Blackjack there is a test that asks questions related to the general programming concepts that we determined were the most important for a foundation in object oriented programming.

The reason that this project is important is twofold. The program we developed can be used to create an interest in computer science early on. Being able to think in terms of objects is very important to a lot of modern day programming, but is often not taught until AP or college level classes. Many existing programs either do not focus on object oriented concepts. The few programs that do provide too much of a disjoint between the objects and the actual lines of code a programmer will eventually be working with. Our program bridges that gap by introducing object oriented thinking while slowly acclimating a user to actual programming languages. Using this program will help a student to develop and cultivate an interest in programming earlier, while providing them with the object oriented foundations that will be necessary as they learn more about computer programming.

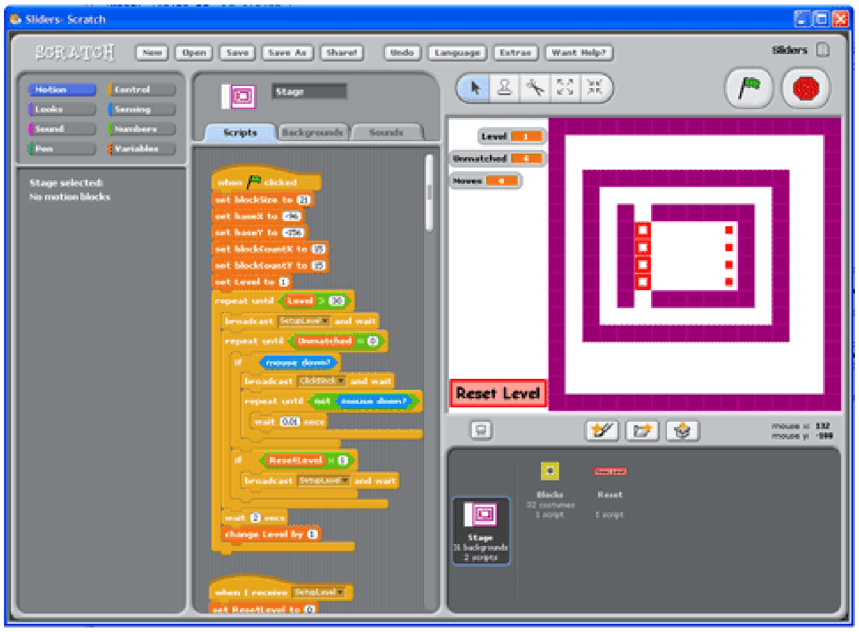
With this project, we designed a program to educate users on the basics of object oriented programming and increase interest in programming concepts. We designed a study to test the effectiveness of our program. The results of the study indicated a average 20% increase in knowledge relating to key areas of programming and object oriented design.

In the remaining portions of the paper we will go further into our background research of the existing field of educational programs. We will explain the different portions that we felt were good additions and the portions that we wanted to approach in a different way. We will also explain the method that we used to build and test the program. After that we will show an analysis of our results. Based on those results we will determine the success of our program as well as postulate changes that may improve how well the program works in reaching our goals.

Background

When we first started the project we began to look into the current state of programs that are being used to teach. We found programs that used their own simplified languages such as Scratch, Greenfoot, Karel, and Small Basic. These programs are all very similar in their approaches. They are all object oriented, but do not actually walk a user through any tasks. In essence, they make programming simpler, but do not actually teach what it is the user is to do. The next group of programs that we found was Robo and Alice. Both of these took graphical approaches, but essentially hid the actual coding portion.

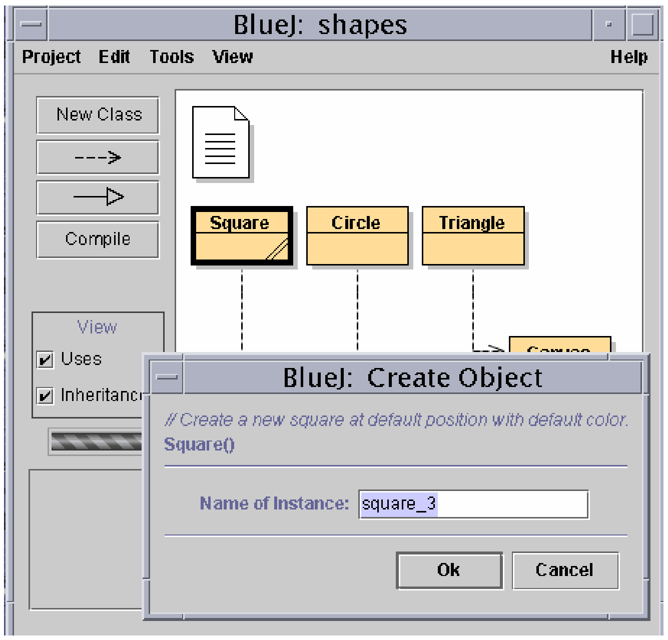
Scratch- This program provided an interesting learning environment. There was a focus on animation to create a program rather than coding. As a result there was a large focus on loops, since that's what many animations use. An interesting thing that we found with this program was the ability to go into the code and change it while the program was running. This allows a very interesting on-the-fly way to code. The main problem with this program compared to the others is that there is not a real focus on the programming portion. (Scratch - Imagine - Program - Share)



```
when [flag] clicked
set blockSize to 21
set baseX to [illegible]
set baseY to [illegible]
set blockCountX to [illegible]
set blockCountY to [illegible]
set Level to 1
repeat until <(Level) > [illegible]>
    broadcast [SetupLevel] and wait
    repeat until <(Unmatched) = 0>
        if <mouse down?>
            broadcast [ClickBlock] and wait
            repeat until <not <mouse down?>>
                wait 0.01 secs
        if <(ResetLevel) = 1>
            broadcast [SetupLevel] and wait
    wait 2 secs
    change Level by 1

when I receive [SetupLevel]
set ResetLevel to 0
```

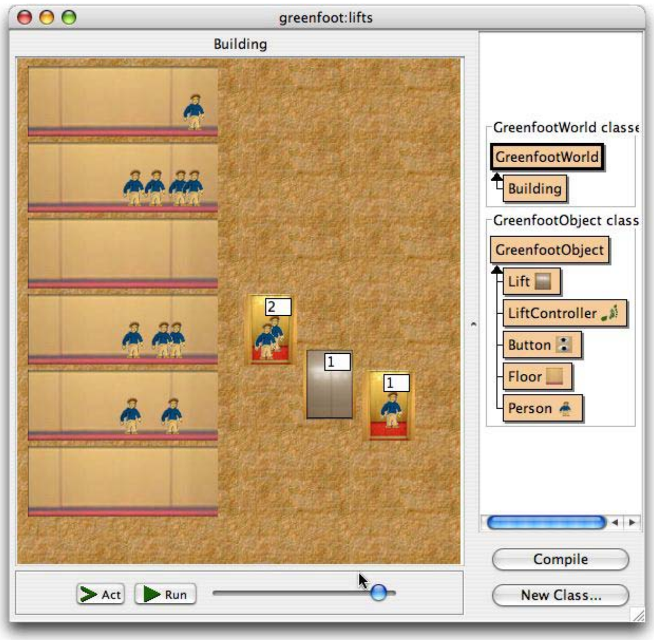
BlueJ- This program used normal java just like actual coding. The difference it made was that it visualized the object oriented design aspect. You would design each part and the visual area would be how to integrate it. Again there was no part that walked the user through what to do. The user was left to read a book to act as the tutorial or have a teacher plan a course using the program. (BlueJ - Teaching Java - Learning Java)



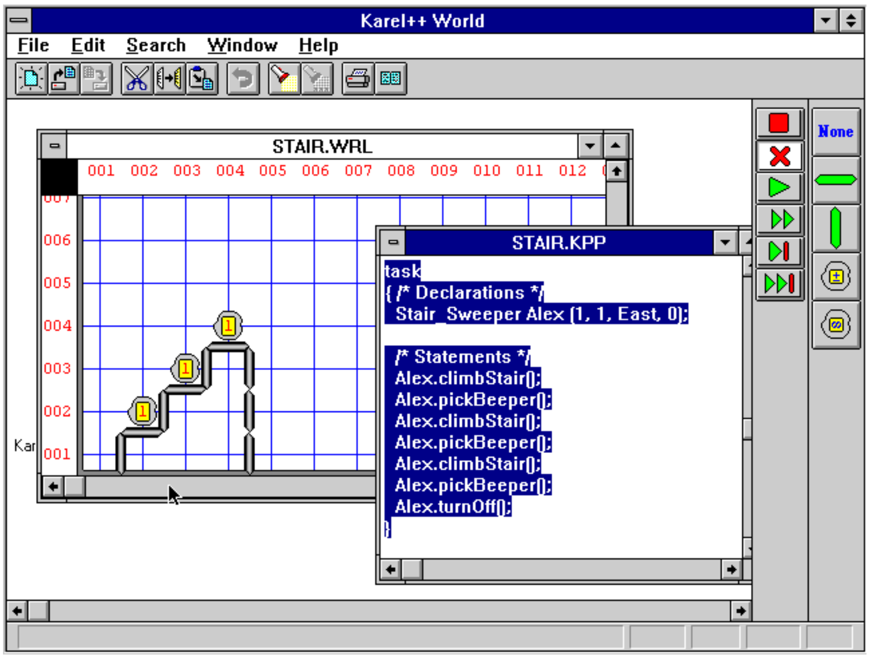
// Create a new square at default position with default color.
Square()

Name of Instance: square_3

Greenfoot- This program we determined to be aimed at intermediate programmers rather than beginners. It provided the most realistic environment of all the programs that we looked at and just used Java as its language. It is clearly not aimed at beginners since it seemed to be more of a last step before actual programming than a way to introduce people. (Greenfoot - The Java Object World)



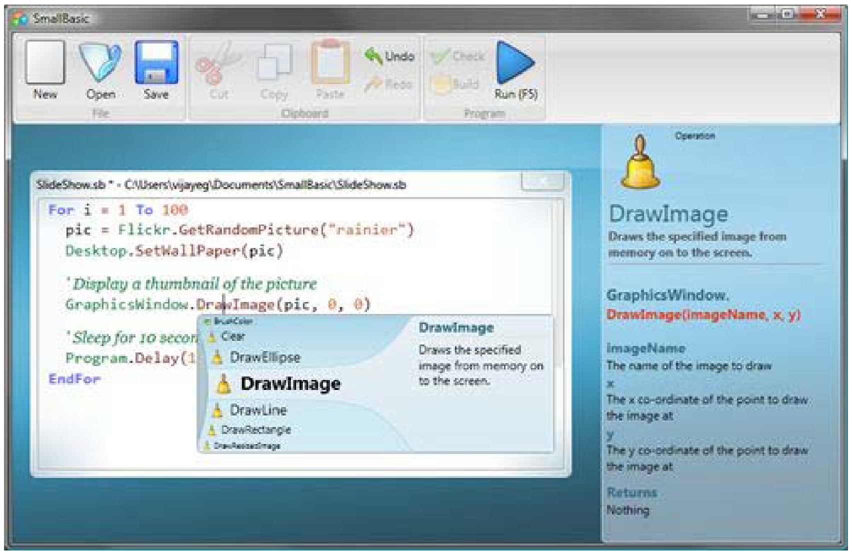
Karel- This program provided many interesting ideas. It used a simplified language and actually highlighted what line of code was being executed. This would let people understand what line of code is causing the problem. The program's idea was simple, you controlled a robot that you gave simple commands; left, right, up, and down. This idea allowed people to quickly learn a lot, but not necessarily anything that is applicable to programming in a real language. (Untch)



```
task
{ /* Declarations */
 Stair_Sweeper Alex (1, 1, East, 0);

 /* Statements */
 Alex.climbStair();
 Alex.pickBeeper();
 Alex.climbStair();
 Alex.pickBeeper();
 Alex.climbStair();
 Alex.pickBeeper();
 Alex.turnOff();
}
```

Small Basic- This program was a very simplified language that was essentially just a coding environment. To understand how to use it there is a large document that explains all the commands and then leaves the user to discover ways to utilize the commands. The part of this program that makes it easy to learn is that it will auto-complete commands for you and provide a description of what that command will do in case you were to forget. (Microsoft Corporation)



SlideShow.sb * - C:\Users\vijayeg\Documents\SmallBasic\SlideShow.sb

```
For i = 1 To 100
  pic = Flickr.GetRandomPicture("rainier")
  Desktop.SetWallPaper(pic)

  'Display a thumbnail of the picture
  GraphicsWindow.DrawImage(pic, 0, 0)

  'Sleep for 10 secon
  Program.Delay(1
EndFor
```

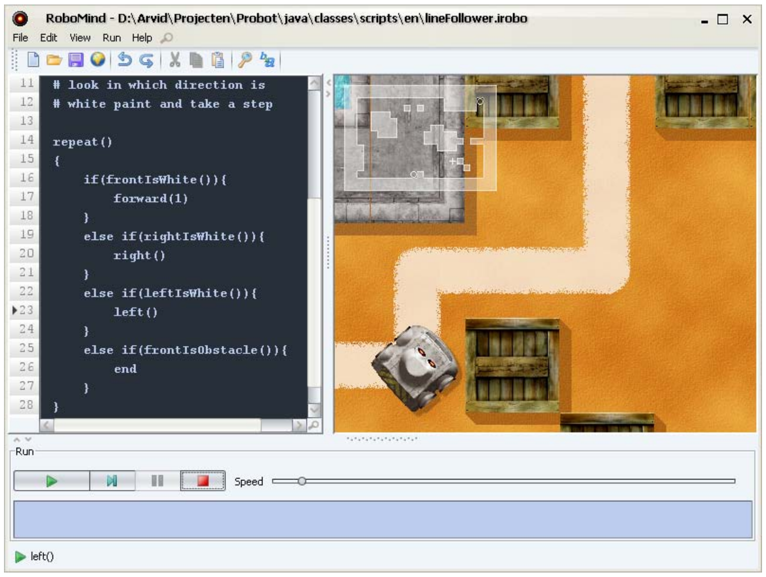
DrawImage
Draws the specified image from memory on to the screen.

DrawImage
Draws the specified image from memory on to the screen.

GraphicsWindow.
DrawImage(imageName, x, y)

imageName
The name of the image to draw
x
The x co-ordinate of the point to draw the image at
y
The y co-ordinate of the point to draw the image at
Returns
Nothing

Robo- Robo is one of the more graphically oriented programs. It is actually very similar to Karel in that you also control a robot on the screen and have the same simple types of commands. The difference is that Robo focuses more heavily on the graphical portion. There are puzzles that people can make and they are forced to program their way out. (Robomind)

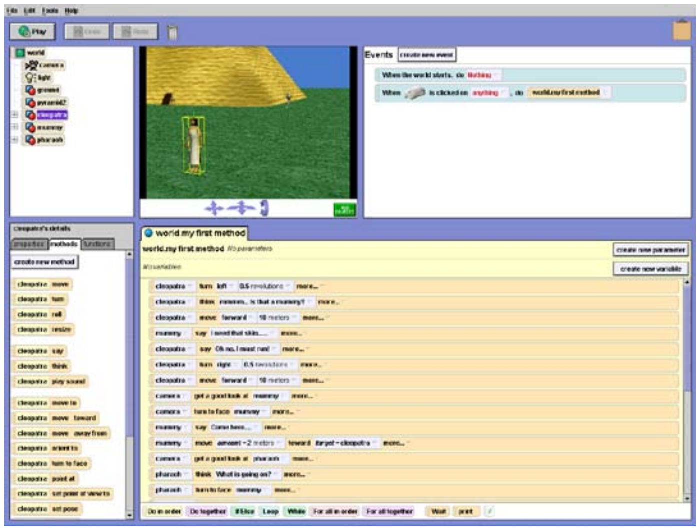

RoboMind - D:\Arvid\Projecten\Probot\java\classes\scripts\en\lineFollower.irobo
File Edit View Run Help
Run
Speed

```
# look in which direction is
# white paint and take a step

repeat()
{
    if(frontIsWhite()){
        forward(1)
    }
    else if(rightIsWhite()){
        right()
    }
    else if(leftIsWhite()){
        left()
    }
    else if(frontIsObstacle()){
        end
    }
}
```

```
left()
```

Alice- Alice is probably the most famous of the “learn to program” programs. It is also the most graphically focused one. Programming is done with 3D images that you drag and drop around the screen. This makes it extremely easy to use, but at the same time limits it. Also when a person moves on to a coding environment they are not really used to seeing text instead of images. (Alice.org)



```
world.my first method No parameters
No variables
cleopatra turn left 0.5 revolutions more...
cleopatra think mmmm... is that a mummy? more...
cleopatra move forward 10 meters more...
mummy say I need that skin..... more...
cleopatra say Oh no, I must run! more...
cleopatra turn right 0.5 revolutions more...
cleopatra move forward 10 meters more...
camera get a good look at mummy more...
camera turn to face mummy more...
mummy say Come here.... more...
mummy move amount = 2 meters toward target = cleopatra more...
camera get a good look at pharaoh more...
pharaoh think What is going on? more...
pharaoh turn to face mummy more...
```

Overall the existing field leaves something to be desired. The offerings will educate and even stimulate the subject matter, but each one on its own isn't really complete. Some don't use real code, which creates a hard learning curve. Others focus on object oriented design to the point where they don't use code at all. Then there are those that are only code and offer very little to emphasize objects and object oriented concepts. There does not seem to be anything that combines real code with ease of use and visuals. Also many of these programs seem to be ones that would assist in a class instead of standalone educational programs.

Some useful papers we looked at were written regarding BlueJ, but also had a lot of general information on effective ways to teach object oriented programming. One paper goes through creating a blackjack program using BlueJ and explaining how it orders the topics. It starts out focusing on the graphical environment since that is the easiest part to understand quickly. It then goes into editing syntax by changing pre-existing code. This offers the idea of typing actual code, but without the challenge of doing it from scratch. There is also the idea that by choosing blackjack as a game people will better understand what is going on. The lesson was able to make connections to ideas in the program with ideas in the actual game. This helps the user learn it more quickly by making these connections. The overall ideas that we got from this paper was that using games that users may already know will likely allow for faster learning as well as making the topic more interesting. The emphasis on starting off with a visual environment also helped us to design our program.

The other paper that we read was mostly out dated, however it did provide a focused idea on how to deal with object oriented design. It talks about it as the main starting point. The paper brings up the idea of starting with giving the user objects that the user is able to move around and interact with to see how everything changes. This focus on object oriented design was much more prevalent than many of the programs we looked at and brought up some interesting ideas. The goal of the paper however was to teach object oriented design to users who already knew some code. In this way it offered an interesting way to approach the problem, but didn't apply directly to what we were trying to do.

Overall there were a few recurring themes in existing programs to teach programming. Many used a graphical approach to simplify things for beginners. The paper that focused on creating a blackjack program corroborates the idea that a graphical approach can be a good idea. The downside to many of the graphical programs was that they often created a significant disjoint between the objects and the actual program code. The ones that did show real code examples, on the other hand, typically had little to no graphical element. The programs focused on one extreme or the other, with none compromising and taking advantage of a graphical interface while providing a solid connection to actual program code.   Programs like Alice used only a graphical environment whereas programs such as Greenfoot only really offered code. While not all these programs are geared towards beginners, there is nonetheless a lack of middle ground. Some approaches used custom programming languages while others used Java. It is interesting noticing the two different approaches.

2. Methodology

Approaching the problem required first determining exactly what capabilities a program would need in order to teach “object first” programming. Before designing a program of our own, we looked at existing programs to see if there were any that fit our purposes or could be easily modified to fit our purposes. We constructed a table of the object oriented concepts that we felt were important, as well as other features a program needed (such as ease of use) and assigned a weight scale to each number. Concepts that were not as important (like the ability to model particularly complex object oriented principles that were beyond the scope of what we were teaching) were rated fairly low, while the ability of a program to represent fundamental object oriented principles was rated very high. We went through each of the programs we were looking at and assigned them numbers according to the weight scale, to figure out which one we wanted to use.

After calculating the values of each program, we looked at the ones which had done best (according to our weighted scale), to figure out whether it was a program we wanted to use. We looked at whether there were any key features we felt were missing, and how difficult it would be to add those features in. We determined that there were no programs that could be modified to fit our needs without significant effort, and that our time would be better spent designing a program of our own from scratch.

Categories	Sub-Categories	Alice	Greenfoot	Karel/Karel++/Karel J Robot	Robo	Scratch	Small Basic	WEIGHT
Usage Tutorial		5	2	1	1	2	1	3
Variables		7	10	0	2	5	10	10
Loops		10	10	10	10	10	10	10
Functions		0	10	10	10	0	10	10
Control Structures		7	10	10	10	8	10	10
Similarity to Actual Code								
	Structure and Code Flow	5	10	7	8	6	10	5
	Commands	1	10	1	6	4	10	2
Flexibility		7	10	4	4	5	10	1
Error Output		0	-5	5	8	0	6	5
Debugging/Stepping		0	2	10	10	6	2	6
UI								
	Complexity	5	10	3	-4	-2	-2	-3
	Intuitivity	5	-7	5	6	6	3	6
	Modifiability	1	2	2	6	2	0	8
Graphical Representation of Code		10	3	8	7	5	2	7
Object Oriented Concepts		6	10	4	0	5	7	6
Actual Teaching (done by *program*)		6	3	6	6	3	0	10
Pre-created World?		5	3	1	8	3	0	4
User Editable World		7	6	7	4	5	0	2
Variety of Scenarios		8	6	2	3	6	6	7
Ease of Editing		1	2	2	6	2	0	10
	Total Values	**578**	**614**	**658**	**805**	**552**	**647**	

Analysis of strengths and weaknesses of existing programs

Looking at possible programming languages to use, the choice was fairly simple. Java is one of the most popular object oriented programming languages, and would provide an excellent foundation for later high school programming classes (most of which are taught in Java, thanks to the AP Exam using Java). We did look at the possibility of using another programming language, but felt it would be easier to design an intuitive program when using a language we were already familiar with – the last thing we wanted was to be learning the same language we were trying to simplify to teach others.

With the programming language decided, we began to construct a script for the teaching part of our program. We looked at existing programming books and tutorials, as well as papers on object oriented teaching, to determine what concepts we wanted to teach (variables, loops, etc.) and the best order to teach them in. While we wanted to give users some hands on experience programming, we also wanted to simplify what they had to do. Since our goal was to teach users to think in an object oriented fashion above traditional linear programming, we wanted the focus to be on objects rather than lines of code. To serve this purpose, we designed our tutorials to only display small sections of code at a time for the user to interact with.

We separated the concepts we wanted to teach into separate chapters, intending each chapter to be a standalone introduction to the topic, and organizing the chapters in the order we felt would best teach object oriented programming concepts. We designed simple examples to demonstrate concepts, and designated certain sections of the tutorial to have interactive code.

One topic that had come up repeatedly in the reading that we had done was the idea that people tend to learn programming better when there are observable results to changes they make. To take advantage of that we decided to design our program so that it would have two primary sections displayed. There would be an area where they learned new programming concepts and applied them, and another area where they could see the results of what they had changed. This kind of visual, instant feedback would help to reinforce the concepts they have learned, as well as help them to think of programming in an object oriented fashion. One of the drawbacks to many current introductory programming courses is that they don’t teach the programmers to think in terms of objects – seeing the graphical results of changes that they make helps users of our program to conceptualize programming in terms of discrete objects that interact with each other.

Since the target demographic of our program was middle and high school aged children, in order to be effective at teaching it was necessary that the program not be boring. If we truly wanted to have a positive impact, the users needed to find the program at least somewhat fun. To this end, it was decided to make the pane of the window where they observe the results of their program into some sort of game. Making it a game would also have the side benefit of further reinforcing the idea of objects, since the user would be able to interact directly with their modified program. We brainstormed possible games that we could have the user be creating and playing, and decided that a game similar in basic concepts to the classic Risk game would be effective. One of the primary reasons we chose it was for the effective way we could tie object oriented concepts into it. With countries that grouped together formed continents, and discrete players, it would make it easy for a user to visualize the way objects can interact with each other and be part of other objects. We simplified the rules of the game down so having to learn complex rules would not detract from the actual programming. Rather than have a user build a program from scratch (we deemed it too daunting a task for a users first exposure to programming), we designed the Risk game ourselves, planning on having the users make modifications and be able to observe the results. Being able to see how the game played out unmodified, and then how it would change when they modified it, would help a user to grasp how the software code they wrote was actually working. We used design patterns so that our Risk game could be adapted to other uses, or expanded easily as our needs expanded. In sections of the code we planned on allowing the user to modify, we made sure to use very clear variable names. When actually constructing our examples, we simplified the code where necessary to make sure the user was only exposed to the code we wanted them to see.

To construct interactive examples, we looked at each chapter and concept and brainstormed existing ways that we could have the user utilize their newfound knowledge of the chapter to make noticeable changes to the Risk game. We then narrowed our results down to the examples we thought would be particularly useful as tests of knowledge, focusing on the ones that could be easily focused on the particular concept we were teaching in its chapter.

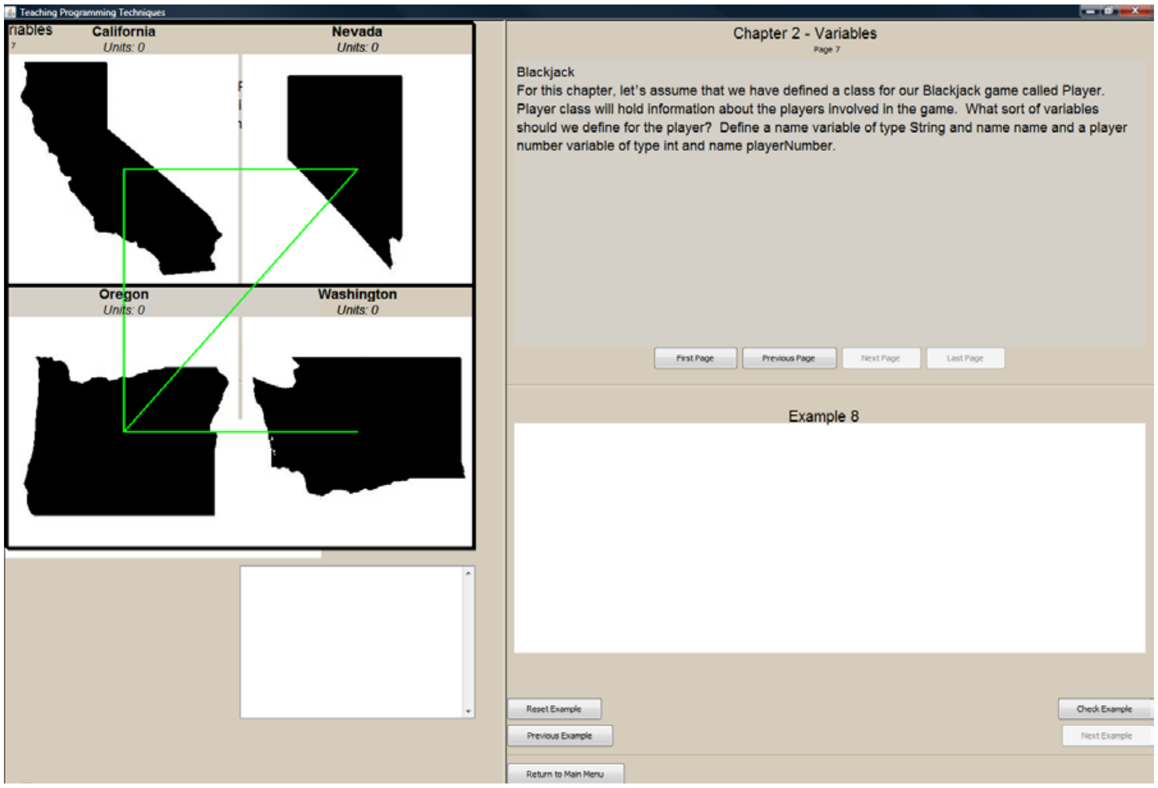
Once we had created our list of examples, it was necessary to determine how we were going to have the user modify code. We thought about using an existing IDE, such as Eclipse or Netbeans, but that introduced a whole new problem. We would need to spend a fair amount of time teaching the user how to use the IDE. On top of that, even if we only wanted the user to be able to use very simple features of the IDE, they would still be presented with a lot of buttons and features they would know nothing about – and when looking at the code would see large chunks of code that was well beyond their current knowledge level, further confusing and overwhelming them. We wanted the focus to be on the objects, not on dozens of lines of code. We determined that the most effective way to allow the user to modify code would be for them to modify it in the program they were already running, where they could see real time results of their modifications in the other pane of the window. To allow their written code to interact with the Risk game, we created hidden code to encapsulate their written code – they would only see and work with a handful of lines of code, but that code was actually surrounded by a complicated program custom tailored for each example to exhaustively analyze their code, figure out what is happening, and then pass parameters into the Risk game indicating what their code was doing. We discussed having their code be compiled directly into the game, but decided that it would be best if errors in their edited code were not able to keep the program from working. Our program wraps the code that they edit in our hidden wrapper code (based on the particular example they are working on), and then compiles it when they select the "check example" button. It tells them if it successfully compiled, and indicates whether or not the code did what the instructions wanted it to do. Whether it does or not, as long as it compiles the Risk game on the left will change to show them the result of their changes. The problem code that they modify is also saved to a temporary variable, so the user can go back to look at other examples or other sections of the chapter without losing any modifications they have made. We also put in a button to reset the example code, in case a user modifies so much that they forget how it started.

Once our program was fully constructed, we wanted to be able to test its effectiveness. We began constructing a series of test questions designed to test a users knowledge of the concepts that the program was designed to teach. We constructed these questions based on our actual program script, as well as examining programming textbooks and online programming courses to see what kind of questions those sources thought were good indicators of how well someone knows object oriented programming.

Once we had test questions, we wanted to find people to help test our program. This was needed for a variety of reasons – the most important was to simply gauge the effectiveness of our teaching techniques to determine whether there were ways we could improve our program. Another, secondary reason was simply to reveal any bugs in the program, or any ways it might not be intuitive. While to us, the program seemed to work fine, and function in an intuitive manner, some of us came from programming backgrounds already, and even the member with little previous programming experience had been working so extensively with the program that he couldn't provide an objective viewpoint. What makes sense to us might not be what a typical user would expect of the program, and as our program was design for people with no programming experience, it was exceptionally important that it seem intuitive to people who have very little or no experience with computer science.

We approached Professor Skorinko about obtaining test participants for our program, and after obtaining the necessary permissions, began to have volunteers test our program to determine its effectiveness. Since we could not guarantee what level of computer science knowledge participants will already have, we have them answer our test questions both before and after they run our program, so that we could compare their answers and look for improvement. We also included open response questions in the second quiz, so that they could explain where they felt the program could be improved or what difficulties they had with the program. One limitation we unfortunately ran into due to time constraints (as well as the difficulty of finding participants willing to participate in a lengthy test) is that our volunteers were restricted to one hour timeslots, which is less time with the program than our original design purposes intended.

3. Results and Analysis



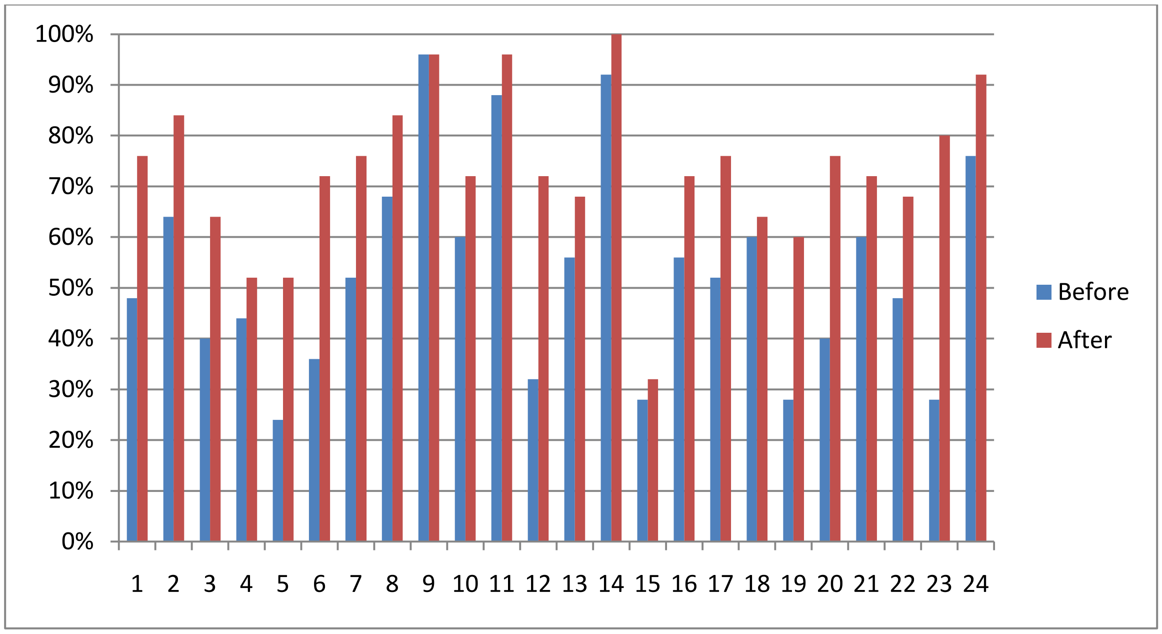
Chapter 2 - Variables

Page 7

Blackjack

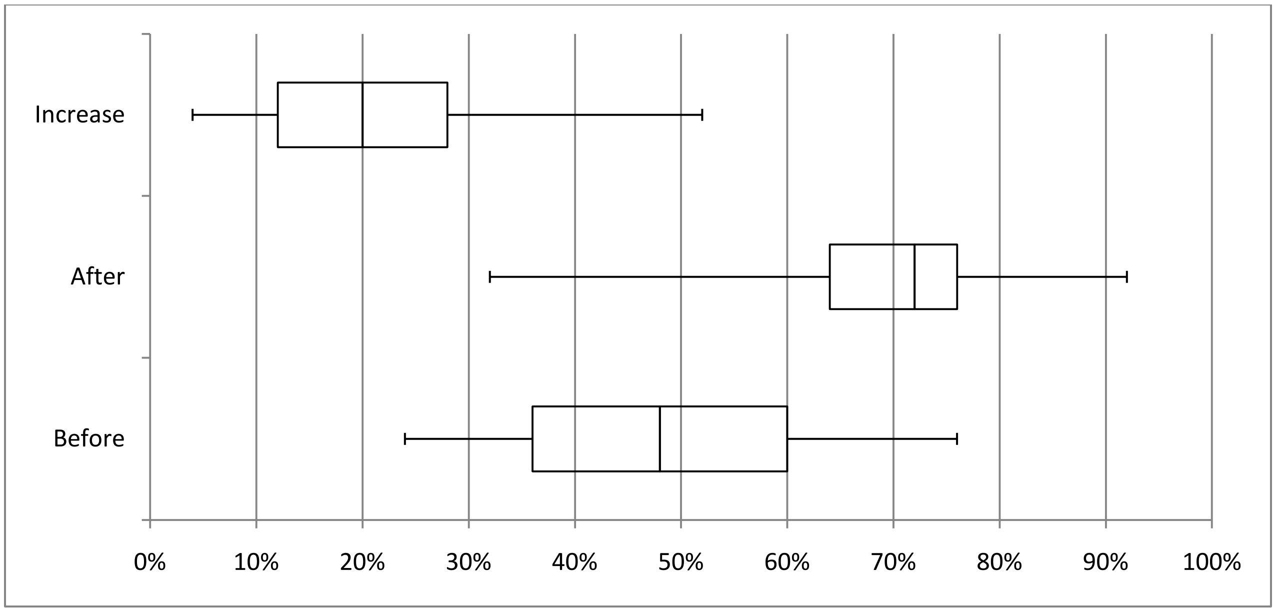
For this chapter, let's assume that we have defined a class for our Blackjack game called Player. Player class will hold information about the players involved in the game. What sort of variables should we define for the player? Define a name variable of type String and name name and a player number variable of type int and name playerNumber.

The final program is different from the original idea. The ideal program was to be a game that taught. The target demographic was also middle school and high school aged children. Overall the program still accomplished the goal of educating and creating an interest as shown in the graph below. We were able to get a total of 25 participants which allowed for a large enough sample size to come up with some general results. It is important to note that we were restricted to approximately 1 hour timeslots to test our volunteer participants, so not all participants were able to run through our entire program.



Analyzing the knowledge scores of the results, the average scores increased 20% after running the program. increase between the test before and after with all the data was 20%. We decided to remove the three users who scored above an 85% on the initial test, since they already had significant prior knowledge and our target demographic is people who don't already know programming. After removing these outliers the average increase in scores was a 22% overall increase. This adjusted data is what we used to work with for all the numbers since it better represents our target demographic.

The starting average of the scores was 48% correct in the initial test, increasing to an average of 70% after running our program. The largest individual result we had was someone who jumped from a score of 28% to 80%, a change of 52%. On the other hand, several individuals only increased their average score 4% after running our program. To show this data a box and whisker plot was constructed.



Looking at the ‘before’ and ‘after’ on the box and whisker chart a noticeable clear increase in people’s scores. While many of the lower scores did not show a significant increase, those who were already scoring in the top quartile showed a substantial increase. An important point to look at is that the middle 50% of test scorers not only increased significantly, but with less deviation in the results – the increases were fairly reliable among people with middle range initial scores. A chart showing the actual numbers of the box and whisker plot is shown below.

	Before	After	Increase
Min	24%	32%	4%
1st Quartile	36%	64%	12%
Median	48%	72%	20%
3rd Quartile	60%	76%	28%
Max	76%	92%	52%

One of the more interesting things that we found was that there was no correlation between the time taken to finish and the change in scores. While most people took the full time that we allowed them to, many took 30 minutes or less and still increased their scores by about 20%.

The rest of the information gained came from the survey section of the test. One issue with our results is that about 13 of the 25 participants had prior knowledge of some kind of programming. While depending on our goals that may seem to invalidate some of our results, it is important to keep in mind that our goal was to teach object oriented concepts. The reason we felt object oriented concepts were an important thing to teach to beginning programmers is because many current curriculums do not focus on objects at all. Subjects who claim to have prior exposure to programming but are still scoring in middling ranges on the test have likely not been exposed to object oriented concepts. Since only three people started off with scores above 85% it would seem like most people's prior knowledge of object based programming was not that extensive – a problem our program is designed to help solve.

People seemed to find each part of the program equally useful. Both RISK and blackjack were overall ranked equally, and the usefulness of our text receiving equal scores from all but one participant.

One of our secondary goals was to help increase interest in programming. We did not receive as positive results here as we had hoped. While no one indicated decreased interest in learning to program, one person did say they were less comfortable with programming than when they started. About half the people felt more comfortable with programming after going through the program, and about a third of the people were more interested in programming. Even some of the people who had prior knowledge before running our program claimed to have an increased interest after the program.

The largest problem participants brought up was the time. Half the people felt that it took too long. There was also a number of people who felt that they needed help to complete some of the examples. These results indicate that our program might work best in an environment where an instructor was there to help walk people through examples, or if the program was improved to better guide the individuals through errors. There were a lot of positive comments in the responses, as well. Many people felt that they really understood the sections that were covered in chapters 1 and 2. The fact that people did not say the same for the later chapters may be a function of the limited time people had to work with our program. Several people indicated that they felt some improvements could be made to the user interface, often in areas that we agreed with.

Overall we were able to increase people’s knowledge of object oriented programming concepts by a significant margin. There was also a measureable increase in computer science interest, though that particular statistic may be somewhat skewed due to the previous exposure to programming many of our participants have had.

Future Work and Conclusions

The original concept for this project was very different from the program we eventually produced. The time constraints that were placed on the project led to us deciding on a different vision of the teaching program. Despite being effective in the goal of teaching programming, the program produced could be improved in several key areas by future work.

The first area that could show improvement is overall presentation. While a simple text interface with a few pictures is sufficient for transmitting information to the learner, it would be more effective with a wider variety of presentation media. More visual learners could be reached by add more visual based textual clues such as coloring and font changes. Adding more pictures and perhaps sound and video would serve to reach a broader variety of learners and also provide more reinforcement for the concepts taught, even to textual learners.

Combined with the improved overall visual presentation, the incentive for learning could be improved. Creating a more entertaining structure for learning could improve learning and retention rates. This could be accomplished with methods such as providing small games which use the learned knowledge to converting completely to a more game-like presentation. The participants in our study were not entertained while using our program, and while that was not the intent of the program, if it is possible to entertain the user without sacrificing teaching utility, we believe that that would be a step forward.

The original vision for the program was a much more game oriented approach, perhaps a top down 2D puzzler with some sort of story to further engage the user. This would be even more important to reach our target demographic of high school and middle school age children as they tend to have a shorter attention span, especially for flat text. Our testing was mainly done on college age students who may have a longer attention span and better retention of purely textual information.

A smaller section that could be improved upon is the program's responses to user input. Some of the examples we provided in the program asked for the user to input a solution and provided a button which would check to see if the answer was correct. This button simply copied the code they provided into a small prebuilt framework of code, compiled it, ran it and looked for correct functionality. While this is usable and many people were able to successfully complete these examples, a better system could be created. Most obviously, incorrect syntax was determined and displayed by the compiler, which is notoriously hard to understand, especially for people who are new to programming. Some method of returning more clear messages and perhaps even hints about how to correct the syntax error would help greatly those who make mistakes of syntax. Mistakes of syntax are common for new programmers especially because of the ease with which they occur. A single misplaced character or capitalization can break an entire program and an inexperienced user would have very little chance of finding the error. In light of this, for the user input examples to be effective, some sort of hint system is almost required. However, in this case, due to time restrictions, we believed that the current system would be acceptable, especially since the user input examples were deemed auxiliary to the learning and would simply supplement the text.

If the user input examples were to be deemed more important, additional problems would also arise. In its current form, the program violently thrusts the user into the syntax of the programming language, in this case Java. While we attempted to provide a lot of similar examples with correct syntax in the hope that the user could learn the key points by repetition, some users in our tests were still having trouble with minor syntactical errors. Future work in easing the user into the syntax of the language or removing syntactical requirements as much as possible would improve the quality of the learning experience, especially for those who are less detail oriented.

In the similar vein of user input, programs such as Scratch have a system whereby the user can create a program and share it with the community. This is an interesting path that could be explored with future work on this project. For example, a user could actually create their own Blackjack game and have others play with it by sharing it with some sort of community, either within the classroom or on a larger scale such as the internet. One of the things that fueled my interest in programming was the ability to easily share my creations with a large audience. Adding the ability for a user to share their creations from within the program may help increase interest in programming and encourage more learning.

One of the things that could definitely be improved on is the experiment itself. We were under a time constraint and had limited people so we could not thoroughly test how effective our program would be with our target demographic. In addition, our initial test contained the same questions as the ending test, which could have biased the user's learning, since they would know what questions they needed answers to. The ideal way to fix this would be to separate our test groups into two distinct groups of people who do not know any programming. One group gets the test without using the program and the second gets the test after using the program. We were not able to get enough subjects to create an experiment like this.

The time slots that were allotted created a large problem for many people. A good portion of people claimed that it took too long, but this was likely a result of the nature of the experiment. The program would be more effective if completed at a users own pace, but people likely felt that they needed to complete the program within the hour, so felt rushed or did not get to take as much time as they needed.

Generally speaking the tests allow us to make several different conclusions. First, the experiment was not as comprehensive as would have been ideal. This was a combination of time constraints and the fact that testing college age students is not necessary a good indicator of how middle and high school aged children would test. Given the fact that many of the college students testing the program felt it could have used someone to help explain examples, it is reasonable to conclude that for a younger age bracket, it would be necessary to have an instructor helping students run the program.

References

(n.d.). Retrieved 11 2008, from Scratch - Imagine - Program - Share: http://scratch.mit.edu

(n.d.). Retrieved 11 2008, from Greenfoot - The Java Object World: http://www.greenfoot.org

(n.d.). Retrieved 11 2008, from BlueJ - Teaching Java - Learning Java: http://www.bluej.org

(n.d.). Retrieved 11 2008, from Robomind: http://www.robomind.net

(n.d.). Retrieved 11 2008, from Alice.org: http://www.alice.org

Arnold, K. (1998). *The Java Programming Language.* Tokyo: Addison-Wesley.

Balagurusamy, E. (1998). *Programming with Java: A Primer.* New Delhi: Tata McGraw-Hill Publishing Company Limited.

Deitel, H. (1999). *Java How to Program.* Upper Saddle River, NJ: Prentice Hall.

Microsoft Corporation. (n.d.). *Small Basic*. Retrieved 11 2008, from MSDN: http://msdn.microsoft.com/en-us/devlabs/cc950524.aspx

Palmer, G. (2003). *Technical Java: Developing Scientific and Engineering Applications.* Upper Saddle River, NJ: Prentice Hall PTR.

Savitch, W. (2006). *Absolute Java.* New York: Pearson Addison Wesley.

Tyma, P. M. (1996). *Java Primer Plus.* Emeryville, CA: The Waite Group Inc.

Untch, R. H. (n.d.). *Karel: Home Page*. Retrieved 11 2008, from http://www.cs.mtsu.edu/~untch/karel/index.html