06B019I

Project number: CEW0702 -51

PRE-COLLEGE CS EDUCATION

An 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

Andrew B. Sutman

Date: December 19, 2006

Approved:

Professor Craig E. Wills, Major Advisor

Table of contents

Abstract

The objective of this project is to identify languages and other such tools designed for introductory programming education, primarily for the purpose of developing new material for use in WPI's Frontiers computer science program. This report discusses why these teaching tools are important, describes some of those available, and explains how one was selected and integrated into the existing Frontiers material.

1. Introduction

The purpose of this project is to identify methods and tools used in introducing young students to programming. Traditionally, students have been taught using full-fledged programming languages from the beginning. Until fairly recently, the best known and most widely used such language was Pascal [1]. Although Pascal was designed specifically for educational use, it was powerful enough to be used in real-world applications, and gradually became popular among professional programmers. The situation today is the reverse: the language most commonly used to teach beginners is Java, which was intended for and is still used in real applications.

The primary motivation for this project was to identify new materials suitable for use in WPI's Frontiers computer science program. In the Frontiers program, high school students live on campus for two weeks, doing daily lab work in one of a variety of fields. The available fields include computer science, mathematics, physics, and biology, among others.

The following section of this report discusses the reasons that specialized tools are preferable to professional languages for educational purposes. The next section presents a small survey of what is actually used to teach programming in high schools in the United States. The remainder of the paper describes the process of adding a new programming environment to the existing Frontiers curriculum.

2. Background

The language most commonly used to introduce students to programming today is Java. C, C++, and Visual Basic are less common, but also widely used. The use of these languages for educational purposes is questionable; they were designed to be used by trained professionals, not rank amateurs. The difficulty of learning to program is compounded by the need to learn to work with professional-grade tools. With no prior experience, students are expected to quickly learn a complex and seemingly arbitrary syntax and numerous special keywords, all so that they can disregard the language itself and learn abstract programming concepts.

A recent study identified a number of problems commonly encountered by beginning programmers. Among these [2]:

- Misusing keywords,
- Omitting keywords or syntactical elements such as braces,
- Differentiating between similar-looking operators, such as = and ==, or subtly different uses of a single operator, and
- Misunderstanding class inheritance.

A major cause of these problems is misleading, inadequate, or nonexistent error messages. Because of the versatility of modern languages, certain errors can cause compilers to misunderstand the programmer's intention and suggest fixing the code in ways that will not produce the desired result. In some cases, programming errors can result in code that is valid. For example, making a statement execute unconditionally when it was not meant to can result from forgetting the keyword "else," and one class can be created inside another if a brace is misplaced. It can be confusing to beginners when code compiles but does not run as expected.

It seems obvious that an introductory programming language should be as easy to learn and use as possible. There are several possible ways of making them so. One of the most effective is to employ a graphical interface for writing code, which prevents typographical and most or all syntactical errors. Some languages are made to be as simple as possible, so that the difficulty of using the language and the number of possible errors is minimized. One environment I encountered, DrScheme, is based on a full-featured language, but restricts its use based on the user's skill level. This allows the interpreter to more accurately determine the programmer's intention and provide more helpful feedback.

In this project, I researched many educational programming tools. I examined how they work, what their creators hoped to achieve, and how effectively they aid education in practice. The following section describes the most interesting ones I encountered.

2.1. Notable languages and environments

2.1.1. Karel and Karel++

Karel is a method of introducing students to programming using a simulation of a simple robot in a simple, grid-based world. Each square in the worlds contains either a wall, a beeper, or nothing. The robot can only move, turn left, and pick up or put down beepers. Karel is intended to teach basic programming concepts using simple constructs, which are represented visually when programs are run [3]. Figure 1 shows an example of a simple Karel world in which the walls are arranged in a square and there are two beepers near the robot.

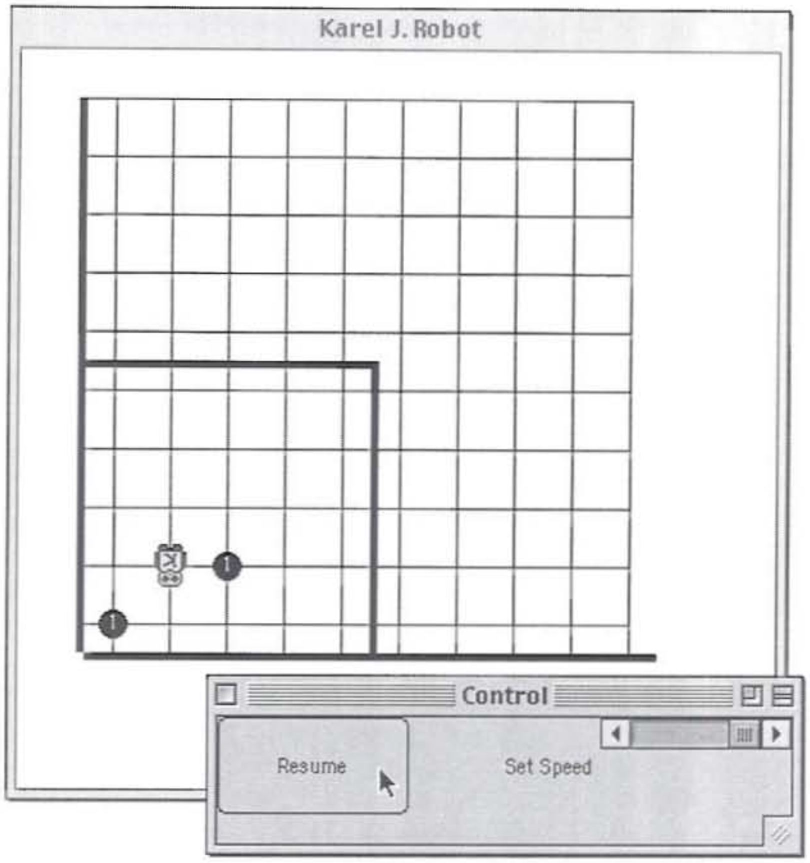


Figure 1: A Java-based implementation of Karel++ [19].

Karel++ is an object-oriented variant of Karel. It extends the concept of the world by adding additional robots. Each robot can be seen as an object, having its own data and methods. One experiment in which two pairs of college-aged students used a Java-based, object-oriented version of Karel reported moderate success. The students involved acquired basic knowledge of object-oriented principles in a short period of time. However, their understanding of these principles, especially that of code reuse, was limited. Also, the two students who had previous experience with procedural languages continued to approach problems in the ways to which they were already accustomed [4].

2.1.2. JPie

Jpie—short for "Java: Programming is Easy"—is a graphical Java programming environment which allows users to write code using drag-and-drop blocks, called "capsules," rather than by typing. This interface is intended to alleviate the difficulty of remembering keywords and syntax while learning the language, as well as to illustrate the similarities and relationships between different pieces of code. JPie uses different icons and colors on its capsules to indicate the nature and use of the code they contain. The creators of JPie hope that by allowing students to manipulate functional blocks of code rather than simply writing text, students will be able to learn programming concepts without having to concern themselves with the language itself [5].

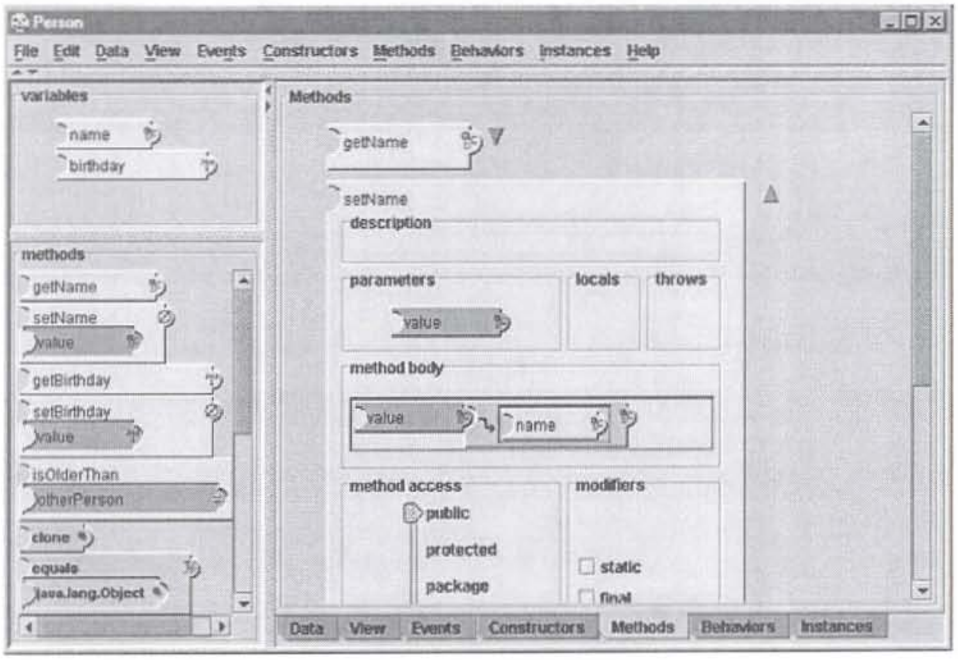


Figure 2: The JPie interface [20].

JPie is currently not well known or widely used. I was unable to find any educators who had used

JPie or any reports from such people of their experiences, so I can say nothing as to how well it works in practice.

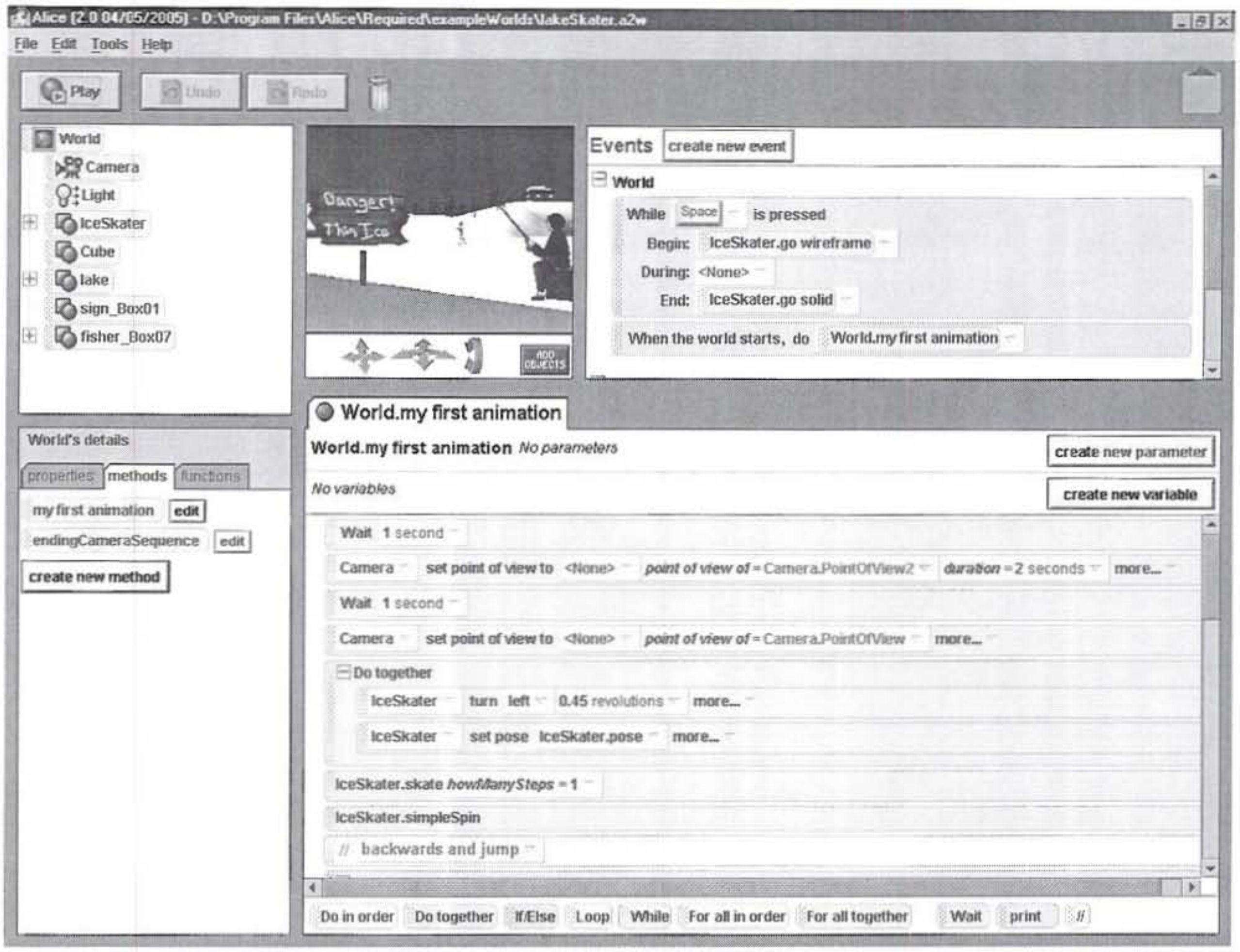
2.1.3. DrScheme

DrScheme is a development environment for Scheme which provides different "levels" of the language for users of different skill levels. By restricting what can be done, DrScheme allows students to learn the language gradually, mastering basic concepts before more advanced ones are even available. This also lets the interpreter give more helpful error messages to beginners, since there are fewer mistakes that can be made and fewer valid ways of correcting them.

DrScheme is extensible, unlike many other learning environments. One noteworthy extension is ProfessorJ, which changes the environment's language to Java. Just as DrScheme does for Scheme, ProfessorJ allows different subsets of Java to be used so that students can be introduced to the language gradually. Currently, however, ProfessorJ is incomplete; it is not fully functional and does not always accept valid code. It is therefore of limited use in actual education for the time being.

2.1.4. Alice

Alice is an introductory programming environment based on the concept of storytelling, developed by the Stage3 Research Group at Carnegie-Mellon University. It uses a world containing 3D objects which can be programmed to move in certain ways and respond to input. Alice uses a unique language which is input mainly via drag-and-drop blocks and drop-down menus. Figure 3 shows most of the features of Alice's interface: a hierarchical list of objects in the world, a list of methods available for one object, the body of one of the methods, code blocks describing how to respond to various events, and the viewport in which the world is displayed.



```
World
  While Space is pressed
    Begin: IceSkater.go wireframe
    During: <None>
    End: IceSkater.go solid
  When the world starts, do World.my first animation
```

```
World.my first animation No parameters
No variables
Wait 1 second
Camera set point of view to <None> point of view of = Camera.PointOfView2 duration = 2 seconds more...
Wait 1 second
Camera set point of view to <None> point of view of = Camera.PointOfView more...
Do together
    IceSkater turn left 0.45 revolutions more...
    IceSkater set pose IceSkater.pose more...
IceSkater.skate howManySteps = 1
IceSkater.simpleSpin
// backwards and jump
```

Figure 3: Defining a method in Alice.

Just as in JPie, the graphical interface prevents typos and eliminates the difficulty of remembering keywords and names. Alice differs in that the language is simpler. While JPie can detect and prevent many errors, Alice lacks many of the features and restrictions that cause these errors in the first place. For example, Alice does not have different data types for integers and floating point numbers or access protection for member data.

In addition to the simple language and graphical interface, Alice features a 3D world which presents object-oriented programming concepts in a intuitive way. The entities in the world represent objects, their actions represent methods, and their properties (position, color, etc.) represent member data. Although many of the more advanced concepts are absent, Alice makes it easier for students to grasp basic concepts quickly.

Alice is also remarkable in that it is designed to be attractive to girls, who are generally less receptive to traditional teaching methods. The creators of Alice believe that the concept of storytelling is more appealing to girls—and often to boys, for that matter—than the more abstract and humdrum concept of programming computers [6].

The effectiveness of Alice as an educational tool was tested at Saint Joseph's University over a two-year period in a project sponsored by the National Science Foundation. Students who took an optional Alice course alongside the standard introductory computer science course performed significantly better than those who did not, especially among those who were considered to be at high risk of dropping out of the CS program. Statistical analysis of the project showed that the high-risk students who took the Alice course averaged a GPA of 2.98 in the regular CS course, compared to 1.18 among those who did not. Of those students, 88% of those who took the Alice course and 15% of those who did not continued to the second CS course [7].

What is described above is Alice 2.0. The original Alice was a completely different program. Version 1.0 was developed by the University of Virginia as a rapid prototyping system for virtual reality software. It did not have its own language; rather, it ran programs written in Python, the same language used to write Alice itself [8]. After Carnegie Mellon University took over the project, it became Alice99, it became an environment for scripting and prototyping the behavior of objects in 3D worlds. There was also a plug-in for Microsoft Internet Explorer and Netscape Navigator allowing users to display Alice worlds in these Web browsers [9].

2.1.5. LEGO Mindstorms

The LEGO Mindstorms series of products allows for the building and programming of robots using standard LEGO bricks. The robots are controlled by a special block, called the Programmable Brick, which houses a processor and a small amount of memory. This block also has a speaker and several connectors for sensors and motors, which serve as the system's means of input and output. LEGO makes four different types of sensors: a light sensor, a sound sensor, a touch sensor, and an ultrasonic distance sensor. In addition to these, there are several unofficial sensors which allow robots to sense ultraviolet light, pH, magnetic fields, and more [10]. Although the only programming languages officially supported are simple, graphical languages, it is possible to use many other languages, including Java, C, Visual Basic, and even Ada [11].

The original Mindstorms system had some constraints which, although not unreasonably strict, do put some limits on its usefulness in education. For one, the original Programmable Brick only had 32 kilobytes of memory. This is not an exceedingly small amount, considering the programs one might want to run, but it does prohibit the use of some particularly complex programs. More importantly, the block has no floating-point hardware, so third-party libraries are needed in order to perform and floating point arithmetic [11]. The current version alleviates the memory restriction somewhat. It has a more powerful processor, twice as much RAM, and 256 KB of flash memory [12]. However, there is still no hardware support for floating-point numbers [13].

Mindstorms was not originally designed specifically for use in formal education. The creators of the Programmable Brick saw it as a way of presenting computers to children as objects that exist and perform actions in the real world, as opposed to a way of interacting with intangible, simulated worlds [14].

2.1.6. Others

I encountered a number of other introductory programming languages and environments in addition to those described above. These are generally less remarkable and not as well known.

A++ is a simple language based on the concepts of abstraction, reference, and synthesis, the three basic concepts of lambda calculus. It is a minimal programming language, billed by its creator as "the smallest programming language in the world." By leaving out all but the most fundamental concepts, A++ is meant to allow students to learn these concepts and gain pattern recognition easily [15].

Although not directly related, the languages Turing and Zeno can both be seen as successors of Pascal. They are designed to be intuitive and use-friendly, so that students do not need to spend much time learning them or dealing with syntax errors. Each language uses a simple syntax apparently derived from Pascal. Each minimizes easy-to-overlook elements such as braces, parentheses, and semicolons, instead relying on more intuitive elements such as words and spaces. In fact, simple programs in the two languages often look almost identical. An object-oriented version of Turing is also available [16, 17].

2.2. Summary

The use of professional-quality programming languages to introduce students to programming is problematic. These languages require knowledge and experience to use effectively, and beginning students have neither. The need to learn a language makes the already difficult learning process that much harder. There are a number of alternatives available in the form of programming languages and environments designed to be easy as possible to learn and use.

3. What is used in high schools today

3.1. Research

In addition to finding what programs are available for introducing students to programming, I looked into what high schools actually use. To do this, I found as many schools as I could which listed this information in their online course catalogs. There were, of course, many schools that did not have course catalogs available on the Internet. In several such cases, I inquired via email as to what, if anything, was used at these schools. I did not receive replies from some schools; the responses I did receive indicated that these schools do not offer programming courses at all.

In all, I found information on 147 schools. Listed below are the languages used in introductory programming courses at these schools and the number of these schools that use each language. Some schools offered more advanced courses using different languages; these were not taken into account. Some schools offer introductory courses in multiple languages, or use more than one language in a single course; these schools are counted once for each language.

- Java: 51
- C++: 28
- Visual Basic: 21
- C: 19
- Scheme: 16
- BASIC: 9
- Python: 6
- JavaScript: 3

3.2. Analysis

Clearly, the use of educational programming software in schools is not as common as might be expected. It appears that a large majority instead use professional languages in introductory programming courses.

It is possible that the creators of these courses were not fully aware of the other options available to them. Although it is not hard to find the software itself, information on its effectiveness is generally more difficult to find.

It seems more likely, however, that specialized educational software was consciously eschewed or never even considered at these schools. One reason these programs might be avoided is because courses using them might not be acceptable for advanced placement purposes. After all, students who learned to use only Karel or Alice might understand basic programming concepts, but they would not have the practical knowledge expected by colleges.

3.3. Summary

Although there are many educational languages available, at least some of which can be shown to improve students' performance in CS courses, it seems that most schools today use only full-featured, often professional languages, most commonly Java, to introduce students to programming. Whether this is because the alternatives are unknown or purposely avoided is uncertain, but the latter seems to be more likely.

4. Frontiers computer science program

The main reason this project was undertaken was to find new materials to use in WPI's Frontiers computer science program, in which students spend two weeks . A unique aspect of the computer science program, and an important consideration in selecting what would be used, is the Frontiers Qualifying Project (FQP). Students are expected to undertake a project demonstrating what they learned, a capstone of their experience in Frontiers.

4.1. Languages already in use

Before this project, there were five lab assignments in the Frontiers CS project: UNIX basics/HTML, JavaScript, Perl/CGI, Java Applets, and StarLogo.

The first of these labs begins by covering basic information on how to use the UNIX-based systems the students would be using for much of their work. After this, it introduces the students to HTML, the language used to make web pages. This is used both to teach interface design principles and to provide the students with a way to view and showcase their later work.

The second lab covers JavaScript, which can be used to make web pages more interactive and dynamic. This, too, is used to teach interface design principles. In addition to this, it is the first true programming language used in the program, and so it is the students' introduction to to many basic concepts such as variables, functions, and flow control.

The third lab covers Perl and CGI. These, like JavaScript, allow more functionality in web pages. This lab focuses more on the technical aspects of programming than the previous one, going into more depth on how the code actually works.

The fourth lab introduces Java applets. Although they are displayed in web pages, these are standalone applications with few limits on their capabilities. The lab assignment walks students through the creation of a simple tic-tac-toe game.

The final lab features StarLogo, an educational programming environment based on the Logo programming language. StarLogo features a simple, 2D world populated by "turtles" which can interact with each other, change colors, and change the colors of the tiles in the world. This lab does not have a structured assignment like the others; it encourages the students simply to explore the software to see what it can do, and then to come up with their own uses for it.

4.2. Deciding what to use

There were several criteria for evaluating the tools available. Ease of use, intuitiveness, and real-world applications were all taken into account. It was also preferable that it be possible to use these new tools as part of an FQP. Most important, however, was that they could be used to effectively teach programming concepts in the short amount of time available.

Although Karel does introduce programming in an interesting way, the concepts it teaches are extremely basic. It is limited to Boolean data, functions, loops, and conditional statements. Because of this, it is not well-suited for Frontiers, as it would not teach much in the short amount of time it would be used. In addition to this, it would be difficult to use Karel in an FQP due to its limited capabilities.

LEGO Mindstorms was a fairly attractive option, as it would allow for the use of any of several languages, and because robots would likely be more interesting than windows and text. The main problem with using the Mindstorms system was the price. The current version, Mindstorms NXT, sells for approximately $250. With an estimated enrollment of 15–20 students, it would be preferable to have at least four sets, which adds up to $1,000. Aside from the price, there would also be the problem of coming up with a suitable activity. Most possible uses for Mindstorms robots, such as navigating mazes, following a path, or wandering freely, would require an area of several square feet for each robot. This would be difficult to provide, due to the limited space available in the laboratory.

Alice seemed to fit all the criteria quite nicely. Its 3D worlds were versatile and interesting, and it did a particularly good job of introducing fundamental programming concepts. Alice also had potential for use in FQPs. Although intended primarily for making animations, it is capable of responding to input. It would therefore be possible to use Alice to make some simple games and applications. A well-made animation might also be a suitable project, even if a fairly simple one.

In the end, seemed to be the most promising prospect by far, and so it was the one and only new program added to the curriculum. With that decided, it fell to me to create a suitable lab assignment for the students to complete.

4.3. Summary

Most of the programming languages and environments described earlier in this report were seriously considered as possible additions to the Frontiers curriculum, But Alice was ultimately the only one selected.

5. The Alice lab

5.1. Creating the lab

Once it was decided that Alice would be used in Frontiers, an appropriate lab assignment had to be created. It was most important to incorporate Alice's object-oriented features, but I also wanted to work in user input as a simple introduction to event-driven programming. The lab would have to be easy enough to be completed by students with no experience within a few hours, but it would be preferable for it not to be trivially simple.

I settled on a simple "whack-a-mole" game. It would use penguins instead of moles, though, as the latter are not among the models included with the software. The game would necessarily be somewhat crude, as Alice's ability to display numbers—in this case, the timer and the score—is extremely limited. There is no way to create text in the 3D world at run time; text is instead displayed in a separate text area. The timer would instead be represented graphically, in the form of a gradually shrinking bar. As for the score, it would be acceptable to print it as plain text after the game ended.

With the goal set, I went through what would be the assignment myself, keeping detailed directions of each step and taking screenshots at key points. From these, I wrote a description of how to complete each part of the process, going into great detail at first and describing each step more generally later on. The complete lab assignment is included as the appendix to this report.

5.2. Results

I served as teaching assistant at Frontiers this year. That was in no way a part of this project, but it allowed me to see firsthand the students' how the students responded to Alice.

Most of the students used Alice long enough to complete the work that was designed for it. Of those, several spent another hour or two afterward doing further work on the assignment or just seeing how much they could do. Most of the more experienced students used it just long enough to determine that the were not interested, or did not even open it.

Three of the fourteen students continued using Alice for a significant period of time after finishing the lab work. One greatly improved upon the design of the whack-a-mole game, adding features to the interface well beyond what Alice was intended to handle. This student, however, was one of the more experienced programmers of the group, and explained that he made the improvements mainly due to a sense of perfectionism. Another made a relatively complex movie, which ran for several minutes.

Out of all the students in attendance, only one used Alice for his FQP, which was a short movie featuring several objects interacting with each other.

The students had mixed opinions about Alice. Most, especially those with little or no programming experience, found it a useful and enjoyable learning tool, as hoped. Others would have preferred to spend more time learning languages with practical applications. Overall, the reception was not as positive as expected, possibly due to the students' average level of prior experience being higher than anticipated.

5.3. Summary

To create the Alice lab assignment, I chose a simple game which would take advantage of most of Alice's interesting features and documented the process of creating it. The students' reactions to Alice were mixed, but generally positive.

6. Conclusions and future work

In this report, I have described the need for specialized educational software for introducing students for programming, as well as several such tools that are already available. I have looked at what is actually used for these purposes in high schools and found that the use of educational software is quite rare, despite its demonstrable advantages.

Existing educational software is being improved and new software is being developed. However, this work amounts to little if the software is never used in the real world. More effort needs to be put into its promotion. It seems this may be done in the future; The next version of Alice is being underwritten by Electronic Arts, the company that developed the popular *Sims* franchise. EA is expected to do much to to promote the use of Alice, possibly to the extent that it will become the new standard for teaching programming [18]. For the time being, however, the development of educational programming tools is largely an academic pursuit.

7. Works cited

[1] Gupta, Diwaker. 2004. What is a Good First Programming Language? 9 Dec. 2006 <http://www.acm.org/crossroads/xrds10-4/firstlang.html>.

[2] Gray, Kathryn E. Towards Customizable Pedagogic Programming Languages. Aug. 2006. 9 October, 2006 <http://www.cs.utah.edu/~kathyg/thesis.pdf>.

[3] Untch, Roland H. "Karel: Fundamentals." 18 Oct. 2006 <http://www.mtsu.edu/~untch/karel/fundamentals.html>.

[4] Borge, Richard, Arne-Kristian Groven, and Annita Fjuk. "Using Karel J collaboratively to facilitate object-oriented learning." Proceedings of the IEEE International Conference on Advanced Learning Technologies 30 Aug.-1 Sep. 2004: 580-584.

[5] Goldman, Kenneth J. "An Interactive Environment for Beginning Java Programmers." Science of Computer Programming 53(1) (Oct. 2004): 3-24.

[6] What is Alice? 2006. Carnegie Mellon University. 7 Dec. 2006 <http://www.alice.org/whatIsAlice.htm>.

[7] Moskal, Bob, Deborah Lurie, and Stephen Cooper. "Evaluating the Effectiveness of a New Instructional Approach." Proceedings of the 35th SIGCSE technical symposium on Computer science education. 2004: 75-79.

[8] Pausch, Randy, et al. "A Brief Architectural Overview of Alice, a Rapid Prototyping System for Virtual Reality." May 1995. IEEE Computer Graphics and Applications. 12 Nov. 2006 <http://www.cs.cmu.edu/~stage3/publications/95/journals/IEEEcomputer/CGandA/paper.html>.

[9] What is Alice? 2000. Carnegie Mellon University. 26 Oct. 2006 <http://web.archive.org/web/20000229152245/http://www.alice.org/about.htm>.

[10] LEGO® NXT Projects. Vernier Software & Technology. 30 Oct. 2006 <http://www.vernier.com/nxt/>.

[11] Klassner, Frank, and Scott D. Anderson. "LEGO MindStorms: Not Just for K-12 Anymore." IEEE Robotics and Automation Magazine Summer 2003.

[12] Lecture notes. Duke University. 7 Nov. 2006 <http://www.cs.duke.edu/courses/fall06/cps097s/notes/lect02-4up.pdf>.

[13] ARM7. ARM Ltd. 7 Nov. 2006 <http://www.arm.com/products/CPUs/families/ARM7Family.html>.

[14] Resnick, M., F. Martin, R. Sargent, and B. Silverman. "Programmable Bricks: Toys to think with." IBM Systems Journal. Sept.-Dec. 1996.

[15] Loczewski, Georg P. 2005. "A++ An Educational Programming Language Based on the Lambda Calculus." 22 Oct. 2006, <http://www.aplusplus.net/>.

[16] Turing Programming Language Home Page. Holt Software Associates Inc. 11 Nov. 2006 <http://www.holtsoft.com/turing/>.

[17] Schmitt, Stephen R. 2006. Programmer's Guide to the Zeno Interpreter. 12 Nov. 2006 <http://home.att.net/~srschmitt/zenoguide/zeno_guide.html>.

[18] Alice press release. 10 March, 2006. Carnegie Mellon University. 17 Dec. 2006 <http://alice.org/simsannounce.html>

[19] 20 Oct. 2006 <http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html>.

[20] 28 Oct. 2006 <http://jpie.cse.wustl.edu>.

Appendix

Objective

Learn some fundamental object-oriented programming concepts using the Alice programming environment.

Activities

Before beginning this project, complete the included tutorials to learn to use Alice's interface, and look at some of the example worlds to see what it can do.

Part 1: The first method

Part 2: Making the penguins appear

Part 3: Multiple penguins and score

Part 4: Starting world.run_game

Part 5: The main game logic

Part 6: Responding to events

Examples

Several example worlds are included with the Alice software. They can be found by clicking the `Examples` tab in the startup dialog. `File -> New` will display this dialog again after it's closed.

Related Links

Alice Home Page

Alice Community

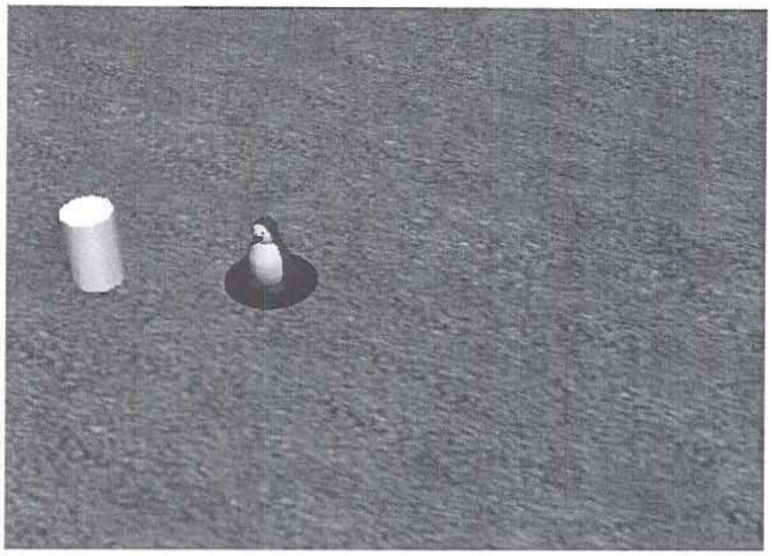
Frontiers Home Page

Part 1: The first method

In this lab, you'll use Alice to create a simple "whack-a-mole" type game with penguins. If you're not familiar with whack-a-mole, you can try a Java version here. To begin, download wam_initial.a2w below and open it in Alice.

wap_initial.a2w

After opening it, the viewport should show something like this:



This world contains one of the penguins, a hole for it to hide in, and a carefully-placed cylinder which will serve as a timer. No methods are ever called, so if you run the world, nothing will happen.

The other penguins will be copied from the existing one. Therefore, it is preferable to make in advance any changes which will be common to all of them; otherwise, they'll have to be made to each penguin individually.

The penguins should begin the game below the ground; if they are moved there now, however, working with them will become more difficult, especially making copies. Instead, you'll tell the penguins to disappear into their holes instantly when the game begins.

Select the penguin and create a new method named `hide`. Drag the penguin's `move` method into definition of `hide`. Select `down`, then `1 meter`. This method will now cause the penguin to move below the ground, but it will take a full second. To make the method run instantly, click `more...` in the `move` block, select `duration`, then `other...`, and enter `0`.

The finished method should look like this.

To test this method, tell it to run when the world starts. Do this by clicking the `create new event` box, selecting `When the world starts`, and dragging `hide` from the penguin's list of methods to the new event's `do` field. Now when you run the world, the penguin should disappear into the ground (you might see the penguin for an instant beforehand; this is not a problem). Delete the event afterward.

Back - Part 6: Responding to events

Next - Part 2: Making the penguins appear

Part 2: Making the penguins appear

Now the penguin needs to be able to move up and down. With the penguin selected, create another new method called `pop_up`. We want this to make the penguin move up one meter, pause, and then move back down over the course of one second.

Much like before, use a `penguin.move` block to make the penguin move up `1 meter`. Then click `more...`, then `duration`, and select `0.25 seconds`. Place a `Wait` block after that, set to `0.5 seconds`. Finally, add another `penguin.move` block which will make the penguin move `down 1 meter` over a period of `0.25 seconds`.

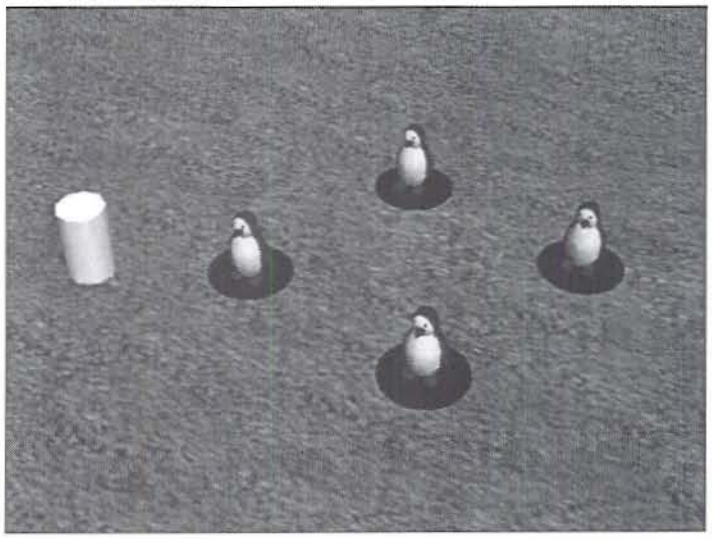
The finished method should look like this.

Back - Part 1: The first method

Next - Part 3: Multiple penguins and score

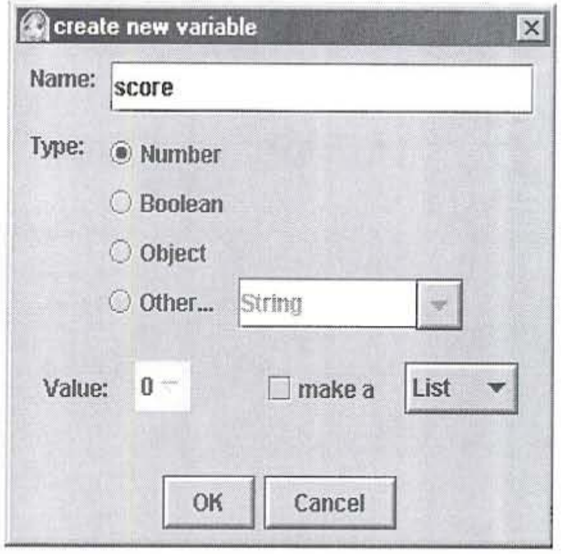
Part 3: Multiple penguins and score

The penguin has all the methods it needs now, so it's time to fill in the board. Click `Add Objects`, then use the `Copy Objects` tool to make three copies of the hole and thee copies of the penguin, so there are four of each in all. Click `done` to return to the programming interface.



If you select the other penguins, you'll notice they have the `hide` and `pop_up` methods you created before.

One more thing is needed before the game logic can be created: a way of keeping score. With the `world` selected, click the `properties` tab and then the `create new variable` button. Name the variable `score`, make its type `number`, and set its value to `0`.



Back - Part 2: Making the penguins appear

Next - Part 4: Starting world.run_game

Part 4: Starting world.run_game

Finally, it's time to create the game logic. Select the `world`, click the `methods` tab, and edit the `run_game` method.

The first thing that needs to be done is to hide all the penguins underground. Drag a `Do together` block into the method from below. Select the first penguin and drag its `pop_up` method into the `Do together` block. Repeat this for penguins 2, 3, and 4.

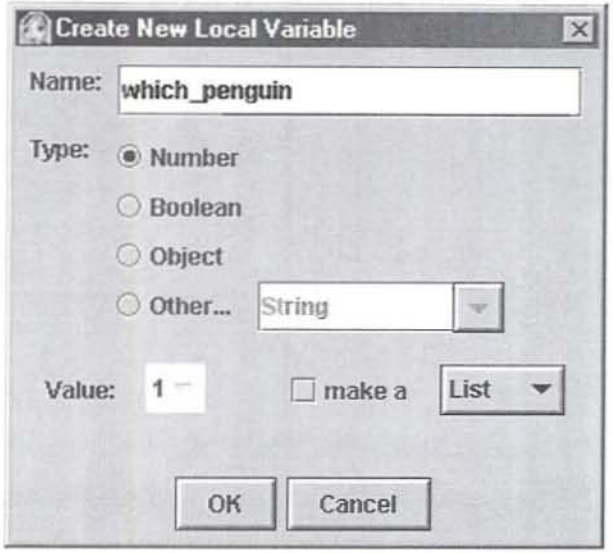
Next, the timer needs to run down, and at the same time the penguins need to pop out of the ground. Add another `Do together` block underneath the first one. Select the `timer` and drag its `move` method into the second `Do together` block. Set it to move `down 1 meter` over a period of `30 seconds`. It should also be set to start and stop suddenly rather than gradually. This is done similarly to setting the duration: click `more...`, then select `style`, then `abruptly`.

Back - Part 3: Multiple penguins and score

Next - Part 5: The main game logic

Part 5: The main game logic

The most complicated part of this method is the logic to select a penguin. Click `create new variable` at the top right of the method definition. This will create a local variable rather than an instance variable. Name this variable `which_penguin` and set its type to `Number`. The value doesn't matter; it will be set later.



The block representing the variable will appear in the top-left corner of the method.

Drag a `Loop` block into the `Do together`, just under `timer.move`. Set the loop to run `30 times`. Drag the `which_penguin` variable into the loop and select `set value`, and then any number. Select the `world`, click the `functions` tab, and drag over `random number` to replace the value you chose to set `which_penguin` to. You'll need to set the values this random number might be. The way this is done is slightly misleading; the number will be equal to or greater than the minimum, but it will always be less than the maximum. Click `more...` on the `random number` block (*not* the `set value to` block), then `minimum`, and then `1`. Set the `maximum` to `5` the same way. You'll also need to set `integerOnly` to `true` so that fractional numbers aren't selected.

Now the method will pick a number from 1 to 4 thirty times. We need a way to use this to choose a penguin. This can be done using nested if statements.

Drag an `If/Else` block into the loop under the `set_value` block. Choose either true or false; it doesn't matter which, as it will be replaced soon. Drag a second `If/Else` block into the `Else` part of the block, and a third into the `Else` part of the second block. At this point, the loop should look about like this.

To determine which statement will be executed, we'll test whether which_penguin is equal to 1, 2, or 3. It's not necessary to test for 4, since that's what it must be if it isn't anything else.

Select the world and click the functions tab. Find a==b under math, and drag it into the first If statement so that it replaces the true or false chosen earlier. Select expressions, then which_penguin, and then 1. Repeat this for the next two If blocks, but with 2 and 3 instead of 1.

The If/Else blocks should now look like this.

The If/Else blocks can now choose what to do based on the value of which_penguin, but so far it doesn't actually do anything. There are four slots which say Do Nothing; these are where the possible actions to take will go.

Select the first penguin and drag its pop_up method into the first Do Nothing slot. Do the same for penguins 2, 3, and 4 in the next three slots. The result should look like this.

The run_game method now chooses a penguin at random and tells it to pop up.

There is one more thing the method needs to do: print the score after the game is over. Place a print block underneath the Do Together blocks. Select expressions, and then world.score.

Finally, this method is complete. It should look something like this.

The methods are all finished now. All that's left is to put them to use.

Back - Part 4: Starting world.run_game

Next - Part 6: Responding to events

Part 6: Responding to events

All the methods the game needs are in place, but none of them are ever actually called. The world needs to respond to some events.

Click create new event and select When the world starts. This will create a new event block. In that block, click the word Nothing and select run_game.

Now the game will run, but it won't respond to input. This will require a few more events.

Create another new event, this time of the type When the mouse is clicked on something. Click anything to select which object should respond to clicks. Choose penguin, and then the entire penguin. To tell the world what it should do when the penguin is clicked, select the world and drag the score variable over the word Nothing in the event block. Alice will ask you what to do with the score; choose to increment it by 1.

This will allow you to score points, but only when you click on penguin 1. Create three more events in the same way for penguins 2, 3, and 4.

When you're finished, the event list should look about like this.

Finally, the game is complete. Click the `Play` button to try it out. Note that Alice's timing is a little odd, so don't be surprised if a couple of penguins show up after the timer has disappeared.

You can also download a finished version of the world below.

wap_final.a2w

Back - Part 5: The main game logic

Next - Part 1: The first method