The Importance of Alice

Curt Hill

Mathematics and Computer Science Department

Valley City State University

101 College St SW

Valley City, ND 58072

Curt.Hill@vcsu.edu

Abstract

There is a dilemma in finding a programming language that is both suitable for instructional use and well supported by industry. There seems to be no suitable language for both purposes. This has led to one language used initially in the introductory Computer Science sequence and another, more popular language later.

One of the candidates for this initial language is Alice, which is exceptionally easy to use and produces engaging output, namely three dimensional animations and games. Moreover Alice uses and demonstrates objects, concurrency and event handling among more common programming language constructs.

The paper concludes with some experiences in using Alice. The first as a recruiting tool before high school students. The second as a quick demonstration of important programming concepts for college students in an introductory class.
The author concludes that Alice has great potential for making programming and also all of Computer Science more accessible to students.

Introduction.

It should be clear that the choice of programming language for a particular task is important to the success of the task. This is as true for the implementation of a project as it is for the education of students. The history of programming languages is strongly influenced by the notion of making the programming process easier. Each new language was designed to make some aspect of the process easier. Assembly language became popular because it was clearly easier to use than the predecessor, machine language. Even FORTRAN, which is one of the most criticized languages ever, made programming easier by using a notation familiar to the people with strong mathematical backgrounds. What an instructor would like in a programming language is a simple, elegant language that may express all the fundamentals of programming, have a good acceptance in industry, be available on a variety of platforms, encourage students to compose good programs and even be affordable. No language satisfies all of these, so what are we to do?

Thirty years ago the choice of Pascal for the introductory programming class was often preferred. Pascal was designed for the education of students, encouraged good programming practice and was widely available. In that day, before the widespread adoption of objects, it demonstrated all the required concepts, usually in an elegant fashion. Prior to that time there was BASIC which may have been the first attempt at a learning language. It was conceived as a simplification of FORTRAN to make it more accessible to college students. There were also subsets such as Ten Statement FORTRAN [Kennedy, 1970] which were also used to assist the learning process.

The programming language Logo [Papert, 1980] needs special mention. The intent was to make the programming process accessible to grade school age children. The syntax was simplified LISP, something like a cross between LISP and an imperative language, so that young students could easily grasp it. Logo also introduced turtle graphics which made the language engaging. Instead of the program producing text on a listing or screen the main product was a graphic, which is much more interesting for the intended age of the students.

Since the heyday of Pascal the choice of the educational programming language has become much more difficult. Pascal has faded in popularity in industry and lacks objects and generics. Pascal's successors, such as Modula-2, Ada, Oberon have not captured the attention that Pascal did or achieve industry acceptance. The C family of languages, including C, C++ and Java, has become the dominant languages in industry [TIOBE, 2006], but there is considerable concern that an "industrial strength language" may be too difficult for the introductory student [Roberts, 2004]. This family satisfies the desire for industry acceptance and wide availability but little else. What are the other candidates? FORTRAN is now considered too specialized. The BASIC family is often convenient, but is a classic example of an inelegant hodge-podge of features. The scripting languages, such as Perl and JavaScript, certainly have their place but are usually too restrictive for

the CS 1 class. The problems go on and on and it seems that the perfect introductory language may no longer be possible.

The idea of using a subset of a language is still common. Object oriented languages such as C++ and Java have the facility to be customized in a way not present in older languages. Easing the use of Java I/O or Graphical User Interface is particularly common, with even the textbooks offering class libraries that simply the language used in the textbook. The C++ overload of the << and >> operators show a move away from the error prone I/O of C as well as an invitation to define classes and operators that make C++ easier to learn.

Some have used a two language approach, with some success. These often involve a simpler language that is used to start the students and then in a subsequent class the students are upgraded to a more serious language. Some of the possibilities include starting with a function language such as LISP or Scheme in the first semester and then introducing C++ or Java in the second. Neither LISP or Scheme could be considered a language that was not serious, but both of these lack industrial support. LISP and Scheme combined have survey usage less than 1% compared to more than 10% support for each of C, C++ and Java [TIOBE, 2006]. Many students have some programming experience before entering their first college computing class and LISP or Scheme opens their eyes to another notation. In addition it familiarizes the students with recursion among other high level concepts.

More recently Python, with an emphasis on using its graphics library to manipulate images, has been used [Guzdial, 2005]. The thinking is that the production of modified images is so engaging that the students are motivated to master the programming concepts. Although this was used in a non-majors course there was a desire to see these students to more seriously consider a computing major. Miller [Miller, 2005] asserts that Python may demonstrate most of the concepts needed, thus it is utilized in the first two courses of the major sequence. Java is then used in the third course. This appears to be similar to the Scheme to Java approach with Python substituted for Scheme.

Is the ideal of a single language that is suitable for both instruction and industrially rigorous applications dead? Perhaps not. What is required is a new language which has the elegance of a Pascal or LISP yet the capability of a Java. This language could be announced tomorrow and popular within a few years. Unfortunately this paper is not that announcement, for right now this language is not to be found.

Alice.

What does the programming language Alice [Dann, 2005] add to this discussion? Actually the question is phrased in a misleading fashion, since Alice is both a language and environment. Removing the language from the environment destroys the beauty of the whole system. Both the language and the environment make it attractive to the educator.

Alice has a number of nice educational advantages, not the least of which is that it is freely available on several platforms [Alice, 2006] since it is implemented in Java. It is inherently a GUI application, the source code is never presented as an ASCII text file, but various pieces with different purposes. The interface is almost purely drag and drop, with very little typing. Since the language and interactive environment were developed together certain frustrations of the beginning student are removed, such as syntax errors. There is no typing of statements, the system will not let a construct be dropped in a way that is illegal. Despite this ease of use, the language clearly demonstrates objects composed of other objects, method calls with required and optional parameters, flow of control, concurrency, event handling and container statements.

Perhaps the most important aspect of Alice is its program output which is a 3D movie or interactive animation. A typical introductory programming class student creates programs that are worthless in the eyes of the student and discarded as soon as they are done. Interesting applications must wait for sufficient experience and knowledge. The typical Alice student may create a movie rather quickly. Student motivation becomes a given rather than something that the instructor has to develop. A bug is not something that needs to be a shown with careful testing but a wrong move in the movie. Alice is easy enough to be useful by Junior High and Middle School age children.

Middle school and older girls are immediately attracted to the ease of creation of a desirable product. This negates the false impression given by aspects of our society that Computer Science is difficult for women. The message works for boys as well in an era of declining CS enrollment.

Alice makes no pretense at being an industrial strength programming language. The main virtues are a very engaging output and an exceptionally short learning curve. The intent of Alice parallels that of Logo. Alice may have a more complicated syntax, yet that is not a problem since it exploits the Graphical User Interface. The development environment must be able to manipulate the statements within the syntax but the programmer does not. The product is also reminiscent of Logo, except the two dimensional line drawings are now replaced by three dimensional animations. Alice is the product of a more mature technology.

These advantages suggested at least two approaches that have been explored. The first is the notion that Alice is a wonderful recruiting tool for Computer Science in pre-college students. The opportunity that appeared was to present Alice in workshops with target audiences of high school students. The second is to use Alice to demonstrate some of the important concepts of programming in a quickened format as the first part of a introductory programming class. The rest of this paper shares some of the experiences of these endeavors.

Camp Cyber Prairie.

Valley City State University has hosted a day camp, named Camp Cyber Prairie for the last several years. The elementary camp is for grades 4 - 6 and lasts two days, the Junior High camp lasts three days and the Senior High camp lasts five days. These occupy two weeks, typically in June. As might be expected from the name there is an emphasis on technology related activities. I was invited to consider offering a session in any of the camps. What was offered was a four hour session on Alice to the Junior High camp and an eight hour session for the Senior High camp.

How much can be done in a four hour workshop? If a person is technically sophisticated, such as a Computer Science instructor, and this person was attending a high level conference like SIGCSE, with a nationally known instructor one might expect to learn enough of the basics of a new programming language in a four hour workshop. This person would not be ready to teach a whole semester class in it yet, but they would have enough to get started and could learn the rest. Now rewind the tape and try again with Junior High students and it should be clear that Alice is not just another language but an exceptionally easy to learn and use language.

In each of the workshops each student had a laptop with Alice installed. I would give a short presentation that showed the basics of what we wanted to do and how to do it. Next would come a demonstration using Alice. Third the students would receive a worksheet that contained a series of steps to accomplish the desired effect. They would get some time to work on their worksheet. The three pieces would take about 30 – 45 minutes to accomplish. The Junior High session lasted about four hours and five such presentation cycles were accomplished. Each of these built upon the previous work. These sessions are described next.

The first presentation was merely how to start Alice and exercise some of the example animations. This was the shortest of the presentations, since the actual time to load Alice and demonstrate an animation and an interactive game is only about 10 minutes.

The second presentation was on the creation of a scene. This was also relatively short. The students choose a background, props and actors that will be in the movie. They learn some of the characteristics of working in a virtual three dimensional world. Then they size and orient the camera, props and actors for the start of the scene. They also learn how to save the file and reload it.

The third session transforms a 3D postcard into a simple animation. In this session they get a taste for object oriented programming. Each action that they can apply to an actor is just a method call on an object. Since the more interesting objects are composed of simpler objects they also learn something about composition. These method calls have required and optional parameters. They must also learn a little about concurrency so that the actors may move at the same time. In Alice the Do-Together statement cause simultaneous actions. It is a statement that contains other statements so without their noticing, they have learned about compound statements. The students learn that a bug is a

program that does not do what we intend. In Alice this is an unintended action of an actor, rather than an incorrect number. They also start testing their programs.

The fourth session is about creating their own methods for existing objects. The prior session demonstrated the use of existing methods, now they create their own. The motivation is easy as well. Making a fish swim is moving the whole fish while at the same time moving its tail from side to side. So wrap all of these actions in a method and give it a parameter that describes how far to move the fish. The creation of a parameter also gives an introduction to the simple types although the workshop mostly wants the default, which is a numeric parameter.

The fifth session introduces the notion of a loop. Like the Do-Together it is another container statement. This could be an infinite loop that keeps the animation running indefinitely or a simple count based loop. They realize that this keeps the length of the code shorter and like all Alice constructs is very easy to add to an existing program.

The Junior High workshop included these five presentations and gave the students time to work on their own animations. Many followed the worksheets that were prepared and others just used the instruction to create movies with different background and actors than I suggested. The important point to realize is that in less than four hours Junior High students, with no previous exposure to Alice, were creating programs with objects, loops and concurrency.

One of the very interesting things about Alice is that by the end of this workshop they have only been required to type three things. They have typed in the name of the file when they saved it originally. They were forced to type in the name of the method that they created. They were also forced to type in the name of the parameter that they added to the method. Every other part of the process was done with just a mouse, clicking menus, dragging objects and the like.

The Senior High workshop had eight hours instead of four. This included the previous presentations as well as several additional ones. These included decisions, event handling, actors carrying other objects, simple variables and some musings on what made a movie interesting. The bulk of the afternoon was devoted to their construction of their own movie.

Since this presentation was mostly about recruiting I sent each student home with a CD which had a full implementation of Alice and links to the Alice web site. I should also note that about half of the students were female in both sessions, in contrast to the normal CS class.

It would also be nice to state that some percentage of these students would eventually attend VCSU or would eventually become Computer Science or related majors, but this paper is about recent experiences, not long term results. It will be very hard to believe that this experience did anything to discourage any of the students from either of these outcomes, since very positive feedback was received.

Alice in the Java class.

How can Alice help an introductory Java class? The class in question, CSci 127, is not the CS 1 class, but an elective that is one way to fulfill a general graduation requirement. This course has no prerequisite and is typically populated by students of diverse majors. Similar to the Python class mentioned earlier [Guzdial, 2005], it will be successful if sufficient students gain a minimal competency, but the desire is also to generate a student interest in further study of Computer Science.

After a successful demonstration of Alice for Junior High students, I became intrigued with the idea of demonstrating the concepts in Alice and before launching into Java. This is exactly what occurred. The first six days (two weeks) of class were used for presentations on Alice. This included all of the presentation material given in Camp Cyber Prairie, as well as some additional information, but was repackaged for a different audience. The students were required to generate one project out of this experience. The enthusiasm for this was not quite as great as the camp presentations but still gratifying. The final presentation pointed out all the features of Alice that corresponded to similar features in Java.

Unfortunately, one cannot add two weeks to the length of a course. Therefore something had to be removed. In the previous version of the course there had been some time spent using the console paradigm. The first programs used a term object that simplified the I/O [Eckel, 1998]. This allowed the first programs to be relatively simple with variables, assignments, arithmetic operators and simple reading and writing of data. The complications of a GUI interface could then be delayed a few programs.

Alice had exposed the students to objects, methods and event handlers in a painless way, so the console material was dropped. In addition, the class used Borland's JBuilder 2005 Foundation package [Borland, 2006] which greatly reduced the typing needed to generate a program. It may also be downloaded for no cost. The GUI is designed in a drag and drop fashion and the typical code the students need to enter is just the code of the event handlers. Even the method header is generated for them. The first windows program which reads a value from an input area, does some calculations and then displays the result in a label is typically in excess of one hundred fifty lines with fewer than a dozen the event handler that was coded by the programmer.

The students seemed to enjoy the detour through Alice, however it is too soon to tell if this ploy will generate more interest in a further study of Computer Science. It is also difficult to tell whether this will amount to a net gain or loss for the course itself. At the date of writing it would seem that this year's class is about one day behind last years. However it is too soon to make an accurate judgment of the consequences of this two week detour.

Conclusions.

This paper presents insufficient data for a conclusion. Instead it highlights two possible uses for an extremely interesting project. In a time when Computer Science enrollment is dropping and the number of women in Computer Science is nearing crisis levels, Alice has much potential for being an effective recruiting tool. However, to be effective it must be demonstrated to the appropriate audience. The day camp was a fortuitous coincidence. The students who would attend such an event as Camp Cyber Prairie are already predisposed towards using computers. What is needed is an effort to demonstrate the ease of use before a wider audience of middle and high school students. This needs to be done before the students, especially the girls, determine that there is no future for them in mastering computer technology.

Exposing college students to Alice in an elective class may well be too little too late with regard to attracting students to the discipline. However, there is also potential in Alice to demonstrate many important concepts in a completely non-threatening way. When the concepts are well understood the students will be more willing to learn the details. Sometimes they never see the concepts for the myriad details. The student who understands the concepts is less likely to doubt that the details can be mastered.

References

Alice. (2006) Alice: Free, Easy, Interactive 3D Graphics for the WWW. http://www.alice.org/index3.html. Date accessed 10 March 2006.

Borland Inc. (2006) JBuilder Download Page. http://www.borland.com/products/downloads/download_jbuilder.html Date accessed 10 March 2006.

Dann, Wanda, Stephen Cooper and Randy Pausch. (2005) Learning to Program with Alice. Pearson – Prentice Hall, Upper Saddle River, New Jersey.

Eckel, Bruce (1998). Thinking in Java. http://mindview.net/Books/TIJ/javabook.html. Date accessed 10 March 2006.

Guzdial, Mark and Andrea Forte (2005). Design Process for a Non-majors Computing Course. In Proceedings of SIGSE 2005, St. Louis MO., (March. 2005).

Kennedy, Michael and Martin B. Solomon (1970). Ten Statement FORTRAN. Prentice Hall, Englewood Cliffs, New Jersey.

Miller, Bradley N. and David L. Ranum, (2005). Teaching an Introductory Computer Science Sequence with Python. Proceedings of the Midwest Instructional Computing Symposium, Eau Claire, Wisconsin. April 8 - 9, 2005.

Papert, Seymour (1980). Mindstorms, Children, Computers, and Powerful Ideas. Basic Books, New York, New York. ISBN 0-465-04827.

Roberts, Eric (2004). The Dream of a Common Language: The Search for Simplicity and Stability in Computer Science Education.

TIOBE (2006). TIOBE Programming Community Index for March 2006. http://www.tiobe.com/tpci.htm. Date accessed 8 March 2006.