3-2007

Determining the Effectiveness of the 3D Alice Programming Environment at the Computer Science I Level

Edward R. Sykes
Sheridan College, ed.sykes@sheridancollege.ca

DETERMINING THE EFFECTIVENESS OF THE 3D ALICE PROGRAMMING ENVIRONMENT AT THE COMPUTER SCIENCE I LEVEL

EDWARD R. SYKES

Sheridan Institute of Technology and Advanced Learning

ABSTRACT

Student retention in Computer Science is becoming a serious concern among Educators in many colleges and universities. Most institutions currently face a significant drop in enrolment in Computer Science. A number of different tools and strategies have emerged to address this problem (e.g., BlueJ, Karel Robot, etc.). Although these tools help to minimize attrition, they have not made significant improvements to this widespread problem. A newcomer to the scene called Alice has been met with positive results by captivating student interest through its rich 3D visual programming environment. During the fall of 2005, Alice, a newly published textbook, and numerous resources were used in Computer Science I at McMaster University. This paper provides an overview of Alice, an assessment of this new course including qualitative surveys, informal observations, and quantitative analysis including student performance score results. Despite numerous technical problems, it was found that the Alice Group exceeded the performance of Comparison Groups: $F(1,93) = 30.322$, $p < .001$ (between C1 and Alice group); $F(1,81) = 4.182$, $p = .044$ (between C2 and Alice Group).

INTRODUCTION

The motivation for changing the manner in which programming is taught in educational institutions comes from the recognition that there has been and continues to be a significant decline in the number of students entering the fields of computer science, computer engineering, and computer programming (Dann, Cooper, & Pausch, 2005; Kessler, 2005; Morris, 2004; Pausch & Conway, 2000; Tucci, 2005; Vegso, 2005). In North America numerous computer departments have reported significant drops in enrolment in their first year computer programs and are struggling to retain even a small number of students in the second and higher year levels (Kessler, 2005; Tucci, 2005; Vegso, 2005). In order to address this problem and to introduce students to a more intuitive "objects-first" approach to programming, Carnegie Mellon University (CMU) developed a richly-interactive 3D graphical programming development environment called Alice (Dann et al., 2005). This visual programming environment offers:

- ease of construction of virtual worlds and situating subtasks to solve in this world;
- a reduction in complexity of details for beginner programmers; and
- visualization of objects in a 3D environment situated in a meaningful context (Dann et al., 2005).

The Alice programming environment provides a means through which students build virtual worlds where objects and their behaviours are situated in a "real" context (Dann et al., 2005). Alice offers the programmer a way to develop realistic 3D animations and programs that support rich interaction with the user (e.g., computer games). A brief description of the Alice environment is described below. Alice is free and is available from http://www.alice.org. Teaching materials are also free and may be found at: http://www.aliceprogramming.net.

As an example of a virtual world that can be created in Alice, consider a problem involving an interactive game in which the user drives a car for a driving test. A scene from this virtual world may look like that presented in Figure 1.

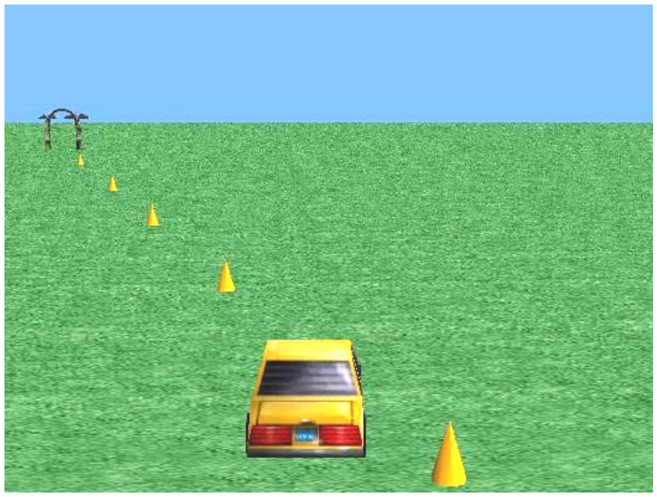


Figure 1. A scene from the driver's test interactive game program.

In order to create and manipulate virtual worlds, Alice provides a Virtual World Editor that has numerous features. Figure 2 depicts the Virtual World Editor. In this editor, students can add 3D objects from a local or internet-based gallery of objects (bottom section) and arrange the position, size, and orientation of each object in the virtual world. Each object encapsulates its own data (private data members such as width, height, and location) and has its own member methods. Students can extend the functionality of objects by adding functions and methods to existing 3D objects (i.e., a form of inheritance). As can be seen in Figures 1 and 2, the Driver's Test program has many objects: a car, several pylons, and a gate representing the finish of the driving test.

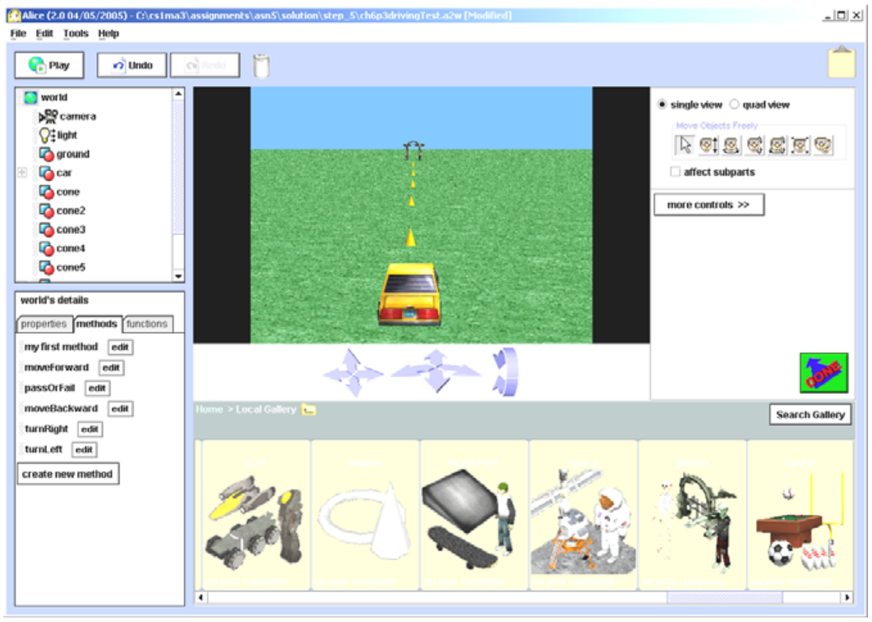
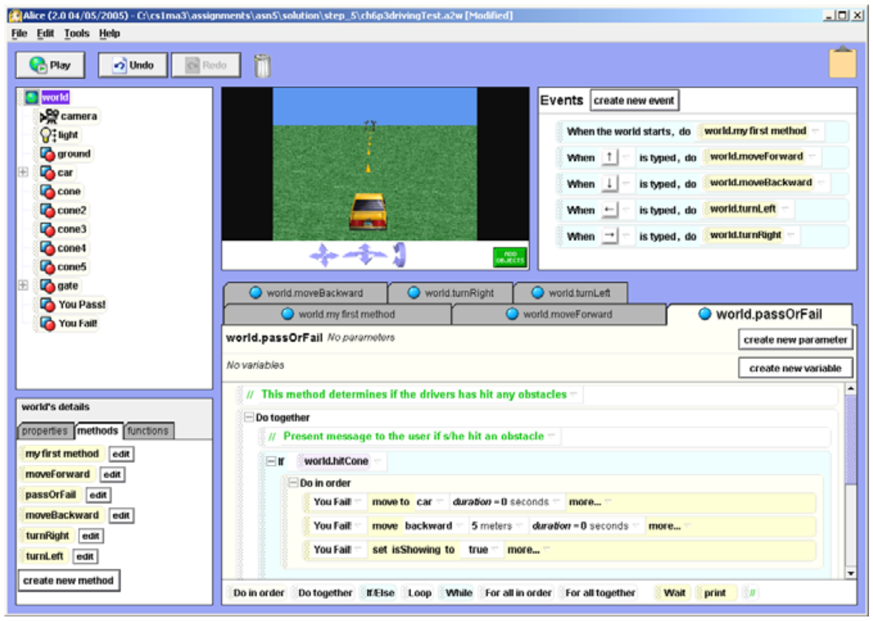


Figure 2. The virtual world editor in Alice.

Once the student has completed the construction of the virtual world, s/he is now ready to "write" the code to support the execution of the interactive game. The Code Editor allows for the logic of the game to be developed using a smart editor. Figure 3 depicts Alice's Code Editor. This editor provides the student a means to create code by extensively using the mouse in a drag-and-drop fashion for predefined objects. To "write" code, the student mouse-clicks an object and drags it into the editor where a context-sensitive drop-down menu appears (Cooper, Dann, & Pausch, 2003). This menu provides options for the student to select from the entire object itself to primitives, subobjects, and functions. Furthermore, the student can write his/her own user-defined functions and methods, which become available in the drop-down menus. As a result, in Alice, "writing code" involves very little traditional typing using the keyboard. Authors of Alice believe this to be a significant benefit as it completely eliminates syntax errors–a major source of frustration among novice programmers in traditional languages such as C/C++ and Java (Cooper et al., 2003; Dann et al., 2005; Moskal, Lurie, & Cooper, 2002).

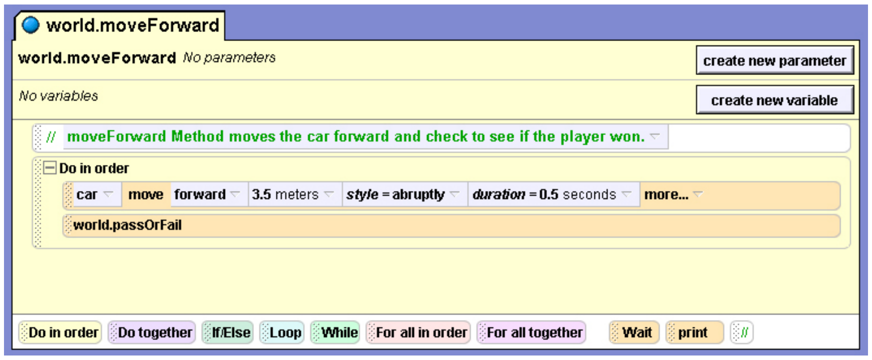
With reference to Figure 3, the components in the Code Editor are the actual code editor (bottom right), world events (top right), an object tree (upper left), and the initial scene for the virtual world (top centre). The tabs in the lower left window allow for querying of properties, methods, and functions relating to the currently selected object. As shown in Figure 3, many user-defined methods were created to support the logic of the driver test game, such as, turnRight, turnLeft, moveForward, and moveBackward. Figure 4 depicts the moveForward user-defined method for moving the car ahead when the ↑ key is pressed by the user playing the game.



```
When the world starts, do   world.my first method
When ↑ is typed, do   world.moveForward
When ↓ is typed, do   world.moveBackward
When ← is typed, do   world.turnLeft
When → is typed, do   world.turnRight
```

```
world.passOrFail  No parameters
No variables
// This method determines if the drivers has hit any obstacles
Do together
    // Present message to the user if s/he hit an obstacle
    If  world.hitCone
        Do in order
            You Fail!  move to  car  duration = 0 seconds  more...
            You Fail!  move  backward  5 meters  duration = 0 seconds  more...
            You Fail!  set  isShowing  to  true  more...
```

Figure 3. Code Editor in Alice.



```
world.moveForward  No parameters
No variables
// moveForward Method moves the car forward and check to see if the player won.
Do in order
    car  move  forward  3.5 meters  style = abruptly  duration = 0.5 seconds  more...
    world.passOrFail
```

Figure 4. moveForward user-defined method to move the car ahead in the driving test game. This method is invoked when the arrow up key (↑) event handler is triggered (see top right component of Figure 3).

LITERATURE REVIEW

This section presents an overview of research that has been conducted in the area of Multimedia Learning in Games and Simulations. Games in education may be conceptualized in two ways: playing educational games (designed and developed by others) or designing your own game. The research literature focusing on whether playing games leads to learning (i.e., increased performance achievement) is mixed (Kirriemuir & McFarlane, 2004; Rieber, 2005). Few differences have been reported in studies involving games used for classroom instruction when compared to traditional class environments (Dempsey, Lucassen, Gilley, & Rasmussen, 1994; Gredler, 2003). However, studies involving gaming in which students learn from designing their own games have revealed promising results. For example, Kafai's research focused on student motivation and learning while building multimedia projects (Kafai, 1994). In these studies, elementary school students were given the task to design an educational game for a younger audience (i.e., grade five students designing games for grade three students). The quantitative findings were increased student performance over comparison groups and the qualitative results showed that students were more focused on class content because the "game design" sessions provided a means for content-related discussions (Kafai, Ching, & Marshall, 1997).

One research study conducted by Rieber, et. al., focused on a qualitative study involving 30 children playing non-commercial games in a classroom environment during a three week period (Rieber, 2005). The following questions were the focus of the study: (a) would children, other then those who designed the game, find these games fun and exciting to play; and (b) based on the children's own play behavior, what features of these games do children report as exemplary? The quantitative study including follow-up interviews with nearly half of the participants yielded interesting findings. The game playing behaviour matched the children's rating of the games. In other words, the games that the children frequently played and played the longest duration were rated most favourably. It was discovered that there are three game characteristics that are important to children; they are: (a) the quality of the storyline; (b) stimulating environment (i.e., visual, auditory sensory stimulation); and (c) suitably challenging (i.e., the game keeps the attention of the user through challenging him/her at an appropriate level of difficulty) (Rieber, 2005). Game-design literature is consistent with these findings (Crawford, 1984; Jonassen, 1994; Rouse, 2004).

Historically, educators have entrusted "computer specialists" with the responsibility of creating tools for instructional purposes. Unfortunately, studies have shown this only constrains learners (Jonassen, 1994; Kafai, 1994; Kafai et al., 1997). In fact, the designers were the only people who significantly benefited from the design process and the use of the tools—not the learners (Jonassen, Wilson, Wang, & Grabinger, 1993; Perkins, 1993). Near the end of the 1990's and early part of this century, the pendulum swung in favour of empowering students by taking the tools away from the instructional designers and giving them to the learners. The consensus was if the design and development tools were in the hands of learners then it would assist in their knowledge construction rather than a tool for conveying and acquiring knowledge (Horwitz, 2005; Jonassen, 1999; Kirriemuir & McFarlane, 2004). The integral part of this philosophy was constructionism—the process of building up the knowledge by using tools would engage learners more fully which would result in more meaningful and transferable knowledge in the learners (Jonassen, 1999; Jonassen et al., 1993; Papert, 1990; Pausch & Conway, 2000; Perkins, 1993; Rieber, 2005).

Numerous studies have been conducted focusing on "learners as designers" and the results are promising: increased student motivation, raised levels of performance, and increased higher-order thought process development (Becker, 2007; Cooper et al., 2003; Dann et al., 2005; Jonassen, 1994; Pausch & Conway, 2000; Robertson & Good, 2004). The Alice programming language is a distinct overlap with these studies since Alice is a tool that empowers the learner to design their own interactive games and multimedia programs while attempting to generate interest in the field of computer science.

METHOD AND PROCEDURES

The method employed in this research focused on determining the effectiveness of Alice as a tool for instruction in Computer Science I courses. In order to determine the degree and quality of learning that took place by students using Alice, a rigorous investigation was conducted using both qualitative and quantitative techniques. The first section of the method is related to the manner in which Alice was evaluated from a qualitative perspective. This research method involved instrumentation including observation, surveys, and personal interviews. The focus of the qualitative investigation was primarily from students' perspectives in the Alice Group (i.e., those students who used Alice for the term); however, input from professors was also gathered.

The second component of the method is related to the manner in which Alice was evaluated based on quantitative analysis of student performance scores. The research method for this section involved a quasi-experimental design with repeated measures. As a result, the researcher was able to compare pre- and posttest performance differences as well group differences (i.e., Comparison versus Alice Group). One advantage of this type of analysis is that interaction effects were able to be calculated and analyzed.

In the quantitative study, the focus was on measuring how much students learned. In support of this objective, construct validity was achieved by: (a) using standardized test theory; and (b) validating the pre- and posttests by asking domain experts to review the tests (Trochim, 2001). Both of these perspectives were accomplished by involving domain experts which included 4 Computer Science faculty members (with speciality in introductory undergraduate level teaching), and 4 Computer Science graduate teaching assistants (who were knowledgeable with the Alice programming language and introductory Computer Science courses). These domain experts reviewed and commented on the content and questions on the pre- and posttests so that appropriate alterations could be made before administering the tests to the students. All tests were a combination of (a) knowledge-based; (b) skill-set-based; and (c) problem solving-based programming problems. In support of standardized test theory, at least half of each test's content were based on high-order thinking skills (i.e., analysis, synthesis, and evaluation) implemented in order to test the students general ability to problem solve (Bloom, 1956; Furst, 1981).

Subjects

The population of this study was students across the province taking a first comparable course in programming. The sample in this study was the students in their first year of university taking an introductory Computer Science I course at McMaster University. Two comparison groups were used in this research. They were students from the Computer Science I course in the summer of 2004 (i.e., Comparison Group 1, [C1]), and the summer of 2005 (i.e., Comparison Group 2, [C2]). The experimental group (i.e., the Alice Group) consisted of the students in Computer Science I during the fall of 2005.

At McMaster University, Computer Science I is considered an "elective" course. As a result, students from various backgrounds and levels took these courses. For example, some students had previously taken several programming courses during high school, while others had not experienced any computer programming at all. Furthermore, the academic level of students in the courses was not consistent. Some of the students were studying Mathematics at the fourth year level, some were in third year of Biology, some were in their fourth year studying Psychology, while the rest were focused on their first year in Computer Science. Nonetheless, these "variations" were consistent throughout all Groups in this study. That is, the same degree of background and level variations were present in both comparison groups and the Alice group.

The Comparison Group 1 consisted of 23 students, the Comparison Group 2 consisted of 11 students, and during the fall of 2005, the Alice Group consisted of 72 students[1]. One professor taught all three groups for the entire term. Both C1 and C2 were taught in a traditional format using the C programming language. For the Alice Group, approximately every week, ½-hour long sessions were conducted by the researcher to elicit specific information about their experience with the course and the Alice programming environment. Additionally, many students posted on the course WebCT discussion forum with comments and suggestions for improvement. The manner in which students were interviewed was primarily individually based; however, there were some

[1] The fall term is the main startup term for all students at McMaster. As a result, the enrollment for this term is the largest. The summer term typically has a much smaller enrollment due to students wishing to transfer into Computer Science or a related program from another program at the university or another university.

occasions when an issue was raised that was a shared concern among several students.

Professors were also selected to participate in this study. The selection of professors was based on a number of factors including their knowledge of 3D programming environments, experience with first year Computer Science, and interest in offering critical opinions on Alice. A total of 4 professors were selected for this study.

Statement of Procedures

Two global procedures were required:

Part A Qualitative investigation on the Alice Group; and
Part B Quantitative investigation on student performance scores.

Part A: Qualitative Investigation on the Alice Group

The research procedure for this section involved a two-phase qualitative investigation that was conducted in the form of surveys during regularly scheduled class periods. The first phase surveys captured general information regarding the course and the use of Alice. This survey was conducted near the beginning of the course. A second survey was issued near the end of the course, after the students had a substantial amount of Alice experience to offer grounded opinions. This survey was an interview-style survey sheet designed to gather specific information from students on their assessment of Alice. The survey included seven open-ended questions to facilitate a great number of perspectives and opinions. Table 1 depicts this measurement instrument. By presenting the survey to students who had used Alice, feedback was gathered representative of both student and professor perspectives. Additionally, the researcher often visited the lab sessions to observe students using Alice. The kind of note taking procedures were observations recorded in a researcher's logbook. Such observations included information regarding individual students' progress through a specific programming problem in Alice. Furthermore, discussions were conducted on a regular basis with several faculty members from the Computing and Software department at McMaster University to elicit opinions from a teacher's perspective of using Alice.

Table 1. Qualitative Survey Sheet

This survey is used to determine the effectiveness of learning within the Alice programming environment. For each question, select the most appropriate response based on the following scale:
1 = strongly favourable to the concept, 2 = somewhat favourable to the concept,
3 = undecided, 4 = somewhat unfavourable to the concept,
5 = strongly unfavourable to the concept.

1. How do you rate the Alice Programming Environment's usefulness?

Very Useful				**Not Useful**
1	**2**	**3**	**4**	**5**

Comments:

____

2. Do you feel Alice is beneficial to your studies? List and explain the advantages/disadvantages of this learning environment.

Very Beneficial				**No Benefits**
1	**2**	**3**	**4**	**5**

Comments:

____

3. Compare Alice with a traditional programming language (e.g., C, Turing, Pascal, etc.). Do you feel Alice is better or worse than these environments? Identify any similarities and differences between Alice and these other programming environments.

Alice is better than other programming environments				**Alice is worse than other programming environments**
1	**2**	**3**	**4**	**5**

Comments:

____

Table 1. (continued)

4. How do you rate the ease with which you use and understand the Alice style of programming?

Very easy to use and understand				**Very difficult to use and understand**
1	**2**	**3**	**4**	**5**

Comments:

____

5. Have you enjoyed Alice? Explain why or why not.

Very Enjoyable				**Not enjoyable**
1	**2**	**3**	**4**	**5**

Comments:

____

6. Do you feel you learn more detailed information in Alice or about the same as a traditional programming language? Explain why or why not.

Learn Better				**Learn the same**
1	**2**	**3**	**4**	**5**

Comments:

____

7. Please add any other comments regarding the Alice programming environment that you would like to share:

____
____
____

Part B: Quantitative Investigation on Student Performance Scores

A series of programming problems were developed for the Comparison Groups and the Alice Group. Students in the Comparison Groups were taught in a traditional format such as instructor-led instruction, group-work, demonstration, etc. using the C programming language. The Alice Group received the same instruction as well but using Alice instead of C. This investigation involved both intragroup and intergroup comparison of student achievement by using pre- and posttest performance tests. Performance tests are small quizzes containing two to four programming problems and space for the student to write their solutions.

The performance tests were administered near the beginning of the term and at midterm. As a result, there were statistical analysis opportunities. These nonsubjective measurements quantify the performance level of students prior to exposure to Alice and allow comparison to the level after exposure to Alice. In addition, comparisons were made between the Alice Group and Comparison Groups. The following section describes the details of the way in which this quantitative investigation procedure was performed.

Prepare a series of programming problems for the Comparison Groups:

1. Select a series of topics that are routinely taught to students when learning the fundamentals of programming, for example, datatypes, identifiers, scope, methods, decision making constructs, and repetition constructs;
2. develop a series of programming problems that are based on those selected topics; and
3. ensure that they meet the requirements of the unit or subunit of study by encouraging several teachers with expertise in this area to review the series of lessons developed.

Prepare a series of programming problems for the Alice Group:

1. Select the same topical area corresponding to the Comparison Group's lessons;
2. develop a series of problems for the Alice Group; and
3. ensure that they meet the requirements of the unit or subunit of study by encouraging several teachers with expertise in this area to review the series of lessons developed in Alice.

Collect data to determine the effectiveness of the learning experience by:

1. conducting the pretest for baseline data on students in the Alice and Comparison Groups prior to exposure to the experiment;
2. determining the mean and standard deviation for the Alice and Comparison Groups;
3. conducting regularly scheduled lectures, labs, and tutorial sessions using Alice to the experimental group;
4. conducting traditional-form lessons for the Comparison groups;
5. conducting the posttest given to both Alice and Comparison Groups; [2]
6. computing standard statistical measures between pre- and postexposure to the two groups respectively (i.e., Alice and Comparison Groups); and
7. computing additional statistical information such as two-way ANOVA with repeated measures.

[2] All tests for this study were knowledge-based and skill-set-based programming problems corresponding to the material covered in the classes.

FINDINGS

The findings of this research are presented in the two respective sections:
Part A Qualitative investigation findings on the Alice Group; and
Part B Quantitative investigation findings on student performance scores.

Part A: Qualitative Investigation Findings on the Alice Group

The culmination of surveys, observations, and researcher's notes were analyzed in an effort to uncover common themes in the students' opinion of Alice. The analysis yielded the following findings:

Student Perspective

The following comments are from students in an effort to uncover common elements regarding benefits and/or problems with using the Alice programming environment.

Positive Comments:

1. "I think it is fun to use Alice, I have used Java in the past (highschool), and there it was all just console based programming. So, with graphics it is easier and more exciting to implement programs in Alice. It has lots of different characters and animations which makes learning much more exciting."
2. "It is really easy to understand the fundamental concepts such as repetition, decision making (e.g., if—else statements), and concurrency in Alice—this is harder to see in other programming environments." [anecdote: The bottom section of Figure 4 contains a list of Alice's programming constructs. This visual display of programming constructs is always available and eases program development for students.]
3. "I love Alice—please keep Alice for future classes! It is enjoyable and fun while learning to program."
4. "Alice gave me an understanding of concepts used in other programming languages."
5. "I liked working with something I could see. Alice is simple to program with drag-and-drop mouse manipulation. It felt pretty rewarding and fun to make the games in this environment."
6. "Alice is good because it focuses on logic (problem solving) rather than syntax."
7. "Alice is really good visually because it allows you to check and run during any stage in the development of your program—just press 'Play' and you can test your program at any time."

8. “Alice is quite helpful in problem solving and its environment makes it easy to break problems into smaller steps.” [anecdote: This is known as the concept of ***stepwise refinement*** discussed in detail in the course.]

Negative Comments:

1. “Slow, unstable, resource heavy, and not very reliable.”
2. “Although Alice is a fun language to use for beginners, it can only do animations and interactive games—it can't do computations like C or Java. As a result, Alice is not suited to programming that solves problems in real world situations.”
3. A number of students felt that the absence of dealing with syntax of a programming language is a disadvantage, as one student stated: “I think you can learn more from a traditional language because you would need to type in the code and deal with both syntax and logic errors. Too much of the work is done for you in Alice with its drag-and-drop coding style.”
4. “Alice is very different in terms of syntax from other languages. Therefore, moving on to a real programming language (e.g., Java) is very difficult for beginners.”
5. Virtually all of the students had experienced Alice crashing:
    a. “Alice has some inherent bugs—in the middle of programming, Alice will crash for no apparent reason.”
    b. “Alice is plagued by a variety of bugs that mask its ability to be a useful tool for teaching. Java could be taught which would be more stable and less frustrating.”
    c. “Alice may be beneficial to someone with no background in programming. However, I spent more time fixing Alice's particular quirks (trying to prevent Alice from crashing, etc.) than actual coding.”
    d. “If there is an error in Alice (i.e., it crashes) it can take a lot longer to fix due to having to reconstruct the method again from scratch because you lose all your work and it doesn’t let you save your code. I prefer to program where I type the code instead.”

Beyond the comments gathered from students, statistical analysis based on the survey was also performed. Table 2 depicts the summary statistics of the qualitative survey from the students’ perspective.

There were a number of interesting observations that result from the analysis of this data. The following are the most significant ones. For question 5, nearly a third of the students (29%) stated that they found Alice to be “very enjoyable.” Furthermore, the statistical measure for the mode was “1,” indicating that the majority of the students found Alice to be "very enjoyable" for this category.

For question 6, only 23% felt that Alice is better than other programming languages. In fact, it was found that 34% of the students felt that other

Table 2. Alice Qualitative Summary Results for Students

Qualitative Summary Results—Students	
1. Usefulness	54%
2. Beneficial	43%
3. Alice is better than other languages	31%
4. Ease of use and understanding of Alice	80%
5. Enjoyable	51%
6. Learn better than in other languages	23%

programming environments are better than Alice. Based on the student comments, this could be a result of Alice's instability, "quirks," and heavy resource demands experienced by many of the students. It leads one to believe that Computer Science I level students would like to learn more established languages such as C/C++, C#, Java, etc. instead of Alice.

Professor Perspective

The professors' perspective largely reflects the same opinions as the students. Table 3 presents the qualitative summary results from the professor perspective. The professors felt the primary concern was the lack of stability in Alice and knowing that the novice programmer may feel the crashes are his/her fault. Numerous tests using Alice (in its original form and the "Slow-and-Steady" versions) were conducted by professors and teaching assistants in an attempt to ascertain why Alice was crashing so often[3]. The types of crashes spanned from unable to save one's work–which could have been numerous hours of work essentially lost, to random errors such as unable to create a user-defined method.

Alice's error output window typically showed a Java stack trace with an unrecoverable message such as a null pointer exception (`NullPointerException`), illegal argument exception (`IllegalArgumentException`), array index out of bounds exception (`ArrayIndexOutOfBoundsException`), etc. Unfortunately, there were no posted solutions to the problems experienced from the main Alice web site,

[3] It should be noted that the machines the students and professors were using exceeded not only the minimum hardware requirements but also the **recommended** hardware requirements (based on information posted on http://www.alice.org).

Table 3. Alice Qualitative Summary Results for Professors

Qualitative Summary Results—Professors	
1. Usefulness	50%
2. Beneficial	50%
3. Alice is better than other languages	25%
4. Ease of use and understanding of Alice	100%
5. Enjoyable	100%
6. Learn better than in other languages	25%

textbook, or other sources in terms of patch updates, FAQs, or forum posts and solutions.

Several professors commented on the structure of the textbook in terms of its treatment of variables. The curriculum coverage of variables was not discussed until near the end of the text (Dann et al., 2005). This is in direct contradiction with the ACM Computing Curricula 2001, numerous other programming textbooks, and current practices in many institutions (ACM Computing Curricula, 2001; (Barnes & Kolling, 2004; Lambert & Obsorne, 2001). The topic of variables (or identifiers), including data members of objects, should be covered early in an introductory programming course since the implementation of algorithms relies on a solid understanding of fundamental concepts of identifiers and the process of storing information during computation (ACM Computing Curricula; (Barnes & Kolling, 2004; Becker, 2007; Jarc, 2004; Lambert & Obsorne, 2001; Salvage, 2001).

Part B: Quantitative Investigation on Student Performance Scores

This section presents the findings of the quantitative investigation of this study. Table 4 presents a summary of the descriptive statistical findings on the performance scores for the two comparison groups and the Alice Group. In order to determine the relationship between the performance scores in C1, C2, and the Alice Groups, a two-way ANOVA with repeated measures was conducted. Table 5 presents the results from the ANOVA for between-subjects effects for C1, C2, and the Alice Group. There was a statistically significant difference between C1, C2, and the Alice Group, $F(2,103) = 16.484$, $p < .001$.

The students in the Alice Group outperformed students in both C1 and C2 Groups. There was a significant level of differentiation between the two Comparison Groups (C1 and C2) and the Alice Group in performance scores. Two 2-way ANOVAs with repeated measures were conducted that confirm these results: $F(1,93) = 30.322$, $p < .001$, indicating there was a significant difference between C1 and Alice groups, $F(1,81) = 4.182$, $p = .044$, indicating a statistically significant difference between the Alice Group and C2 at the 0.05 level. Tables 6 and 7 show the results from these ANOVAs.

Table 4. Standard Statistical Measures for C1, C2, and Alice Groups

Group	**Pretest** mean and *(standard deviation)*	**Posttest** mean and *(standard deviation)*
C1	56.308 *(12.279)*	61.908 *(18.312)*
C2	66.863 *(8.506)*	69.318 *(9.306)*
Alice	71.708 *(13.741)*	80.096 *(15.008)*

Table 5. Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C1, C2, and Alice Groups

Source	**Type III sum of squares**	***df***	**Mean square**	***F***	**Sig.**
Intercept	556397.917	1	556397.917	1814.578	.000
Group	10108.625	2	5054.312	16.484	.000
Error	31582.549	103	306.627		

Table 6. Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C1 and Alice Groups

Source	**Type III sum of squares**	***df***	**Mean square**	***F***	**Sig.**
Intercept	635482.520	1	635482.520	1959.822	.000
Group	9832.034	1	9832.034	30.322	.000
Error	30155.731	93	324.255		

Table 7. Two-way ANOVA with Repeated Measures: Between-Subjects Effects for C2 and Alice Groups

Source	Type III sum of squares	*df*	Mean square	*F*	Sig.
Intercept	395694.305	1	395694.305	1421.174	.000
Group	1164.420	1	1164.420	4.182	.044
Error	22552.643	81	278.428		

Figure 5 shows a pictorial summary of performance scores between C1, C2, and the Alice Group using the mean grades as the data. It is evident that the Alice Group posttest performance was significantly higher than both comparison groups. The Alice Group (80% at posttest) finished a grade letter above C2 (69% at posttest) and nearly two letter grades above C1 (61% at posttest).

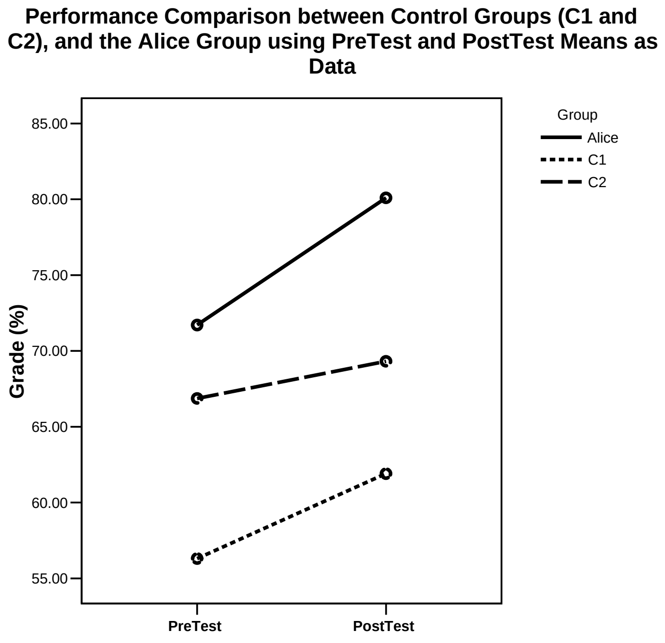


Figure 5. Alice Group versus C1 and C2 performance comparison using pretest and posttest means as data.

CONCLUSIONS

The goal of this study focused on determining the effectiveness of Alice as a tool for instruction in Computer Science I courses. In order to determine the degree and quality of learning that took place by students using Alice, a rigorous investigation involving qualitative and quantitative techniques was conducted.

The results from the first section of the study investigated the use of Alice from a student perspective using qualitative instruments. The results from this section of the study distinctly fall into two main categories–those that support the use of Alice, and those that disfavour Alice in Computer Science I courses. The benefits of using Alice included:

- Alice completely eliminates syntax errors which is one of the most problematic concepts for beginning programmers (Dann et al., 2005).
- Alice is a highly visual programming environment that allows students to create animations and interactive games in a fun, easy, and exciting way, which is very attractive for entry-level college and university students.
- Alice enables students to focus on problem-solving skills instead of spending time on syntax errors, compilation errors, and environment setup problems (e.g., as in Java).
- Overall, students found Alice to be enjoyable (51%), easy to use and understand (80%), and beneficial (43%).

The disadvantages of Alice are summarized below.

- The current version of Alice has a number of problems relating to stability. The main problems were (a) random unexplained crashes (where one's work is completely lost), (b) inability to save one's work, and (c) irreconcilable Java stack trace errors (where a popup window would appear and the "ignore" and/or "retry" buttons would have no effect).
- Not all of the students were happy with a "game programming environment." A number of students stated they would have been happier with a traditional programming environment to develop core programming skills, including skills and knowledge of debugging syntax errors.
- Alice programs often led to long, verbose code that was hard to read and understand.
- Alice is not suited to solve "real-world" problems. Languages such as Java, C/C++, and FORTRAN are general-purpose languages that are used to solve a myriad of different types of "real-world" problems (Arnow & Weiss, 2001; Chapman, 2003; Deitel, 2002; Gosling, Joy, Steele, & Bracha, 2000; Kernighan & Ritchie, 1988; Stroustrap, 2004).
- Students do not develop any skills in typing code, resolving syntax, compilation, or environment setup problems—problems that exist in virtually all other programming languages.

The second section of this study involved an investigation of student performance scores. Three classes were involved in this study, the Alice Group and two comparison groups. In all of the experiments, the Alice Group significantly exceeded the performance of the Comparison Groups, C1 and C2. Two two-way ANOVAs with repeated measures were conducted that confirm these results: $F(1,93) = 30.322$, $p < .001$, indicating there was a significant difference between C1 and Alice groups, $F(1,81) = 4.182$, $p = .044$, indicating a statistically significant difference between the C2 and the Alice Group at the 0.05 level. These results, coupled with the generally positive qualitative feedback from students and professors, indicate that Alice is a good environment for novice programming students. It is interesting to note that, despite the numerous technical problems, the Alice group persevered and rose over these problems to outperform the comparison groups. The findings from the qualitative aspect of this study speak to this issue. Students spent up to four times more time on the Alice course over other courses they were taking in the same semester. Students found the environment fun yet at times, quite frustrating. In many respects, it is very similar to many computer games that are addictive yet at the same time frustrating when you lose (Gredler, 2003). This addiction keeps the attention of the student and stimulates him/her to do better. Research literature in this area support this perspective (Gredler, 2003; Horwitz, 2005; Jarc, 2004; Kafai, 1994; Moskal et al., 2002; Rieber, 2005; Salvage, 2001; White & Frederiksen, 2000).

Recent investigation of the number of students taking Computer Science I has increased significantly for the fall 2006 term (the year following the introduction of Alice to the curriculum). The fall 2006 Computer Science I group at McMaster University had 105 students—a 33% increase over the 2005 group. This could be attributed to a number of reasons, however, it is reasonable to speculate that the word got out that the course is enjoyable and uses an interactive and game-like programming environment.

FUTURE RESEARCH

A number of questions have been raised by the current study that future research could examine. First and foremost is the problem of Alice's stability. In order for Alice to be successful in an environment of novice programmers, it must be stable. In this research a number of students stated the difficulties they experienced due to Alice errors and Alice crashing. Future development of Alice must improve on this aspect and perform significant testing before releasing it to the public as a fully developed version.

Furthermore, there is the question of transfer: How are students performing in Computer Science II? Currently, research is underway to gather information about this question: "After learning introductory programming in Alice, are students able to program in a more traditional language with ease or with great difficulty?"

It was recently announced (March 2006) that Alice has commenced a complete overhaul that will span the next 18 to 24 months (Watzman & Spice, 2006). Electronic Arts Incorporated (EA) has agreed to help underwrite the development of Alice 3.0 with Carnegie Mellon University with the primary goal of dramatically improving the degree of "realism" in the 3D characters and animation in Alice. EA designed and developed "The Sims™", a very popular PC video game. On the Alice web site, "Experts say that when the transformation is complete, the new programming environment will be in position to become the national standard for teaching software programming" (Watzman & Spice, 2006).

While this redevelopment of Alice to improve on its "visual" features may be exciting and beneficial for learners, managers and developers of version 3.0 need to address how Alice may be improved in terms of its stability. Furthermore, if there were a seamless approach to bridging Alice's code-style to other "conventional" programming languages, such as C/C++ or Java, it would ease the transition for students into environments in which syntax, compilation errors become an issue. Ultimately, the goal for all first year Computer Science educators is to provide an interesting, rewarding experience and to increase the chances that students will continue and succeed in Computer Science. With some careful adjustments, Alice may be the means by which educators may accomplish this goal.

REFERENCES

ACM computing curricula 2001. (2001). *ACM Special Interest Group in Computer Science Education,* Vol. *3,* p. 267-298.

Arnow, D., & Weiss, G. (2001). *Introduction to programming using Java: an object-oriented approach*: Addison-Wesley.

Barnes, D., & Kolling, M. (2004). *Object First with Java -- A Practical Introduction using BlueJ* (2nd ed.): Prentice Hall / Pearson Education.

Becker, B. W. (2007). *Java: Learning to Program with Robots*: Thomson Course Technology.

Bloom, B. (1956). *Taxonomy of educational objectives: the classification of educational goals. Handbook 1: Cognitive domain, .* New York: New York, McKay.

Chapman, S. J. (2003). *Fortran 90/95 for Scientists and Engineers*: McGraw-Hill.

Cooper, S., Dann, W., & Pausch, R. (2003). *Teaching Objects-first In Introductory Computer Science.* Paper presented at the 34th SIGCSE technical symposium on Computer science education, Reno, NV, USA.

Crawford, C. (1984). *The Art of Computer Game Design*. Berkeley, Calif.: Osborne/McGraw-Hill.

Dann, W., Cooper, S., & Pausch, R. (2005). *Learning to Program with Alice*: Prentice Hall.

Deitel, D. (2002). *C++ How To Program* (4th ed.): McGraw-Hill.

Dempsey, J., Lucassen, B., Gilley, W., & Rasmussen, K. (1994). Since Malone's theory of intrinsically motivating instruction: What's the score in the gaming literature? . *Journal of Educational Technology Systems, 22*(2), 173-183.

Furst, E. (1981). Bloom's Taxonomy of Educational Objectives for the Cognitive Domain: Philosophical and Educational Issues *Review of Educational Research, 51*(4), 441-453.

Gosling, J., Joy, B., Steele, G., & Bracha, G. (2000). *The Java language specification* (2nd ed.): Addison-Wesley.

Gredler, M. E. (2003). Games and simulations and their relationships to learning. In D. H. Jonassen (Ed.), *Handbook of research for educational communications and technology* (pp. 571-581). Mahwah, NJ: Lawrence Erlbaum Associates.

Horwitz, P. (2005). *GenScope Project*. Retrieved October 20, 2006, from http://genscope.concord.org/about/staff.html

Jarc, D. (2004). *Computer Science I/II, C++ Interactive Exercises*. Retrieved March 20, 2006, from http://nova.umuc.edu/~jarc/sdsd/

Jonassen, D. H. (1994). *Technology as Cognitive Tools: Learners as Designers*. Retrieved November 3, 2006, from http://itech1.coe.uga.edu/itforum/paper1/paper1.html

Jonassen, D. H. (1999). *Computers as Mindtools for Schools: Engaging Critical Thinking* (2nd ed.): Prentice Hall.

Jonassen, D. H., Wilson, B., Wang, S., & Grabinger, R. (1993). Constructivist uses of expert systems to support learning. *Journal of Computer-Based Instruction, 20*(3), 86-94.

Kafai, Y. (1994). Electronic play worlds: Children's construction of video games. In Y. Kafai & M. Resnick (Eds.), *Constructionism in practice: Rethinking the roles of technology in learning*. Mahwah, NJ: Lawrence Erlbaum Associates.

Kafai, Y., Ching, C., & Marshall, S. (1997). Children as designers of educational multimedia software. *Computers and Education, 29*, 117-126.

Kernighan, B., & Ritchie, D. (1988). *C Programming Language (2nd Edition)*. New York, New York: Prentice Hall.

Kessler, M. (2005). *Fewer Students Major in Computer Science*. Retrieved March 20, 2006, from http://oracle.ittoolbox.com/news/display.asp?i=129596

Kirriemuir, J., & McFarlane, A. (2004). *Literature review in games and learning: A report for NESTA Futurelab*. Retrieved November 3, 2006, from http://www.nestafuturelab.org/research/reviews/08_01.htm

Lambert, K., & Obsorne, M. (2001). *Java: A Framework for Programming and Problem Solving* (2nd ed.): Brooks/Cole.

Morris, J. (2004). *Programming Doesn't Begin to Define Computer Science*. Retrieved March 20, 2006, from http://www.post-gazette.com/pg/04186/341012.stm

Moskal, B., Lurie, D., & Cooper, S. (2002). *Evaluating the Effectiveness of a New Instructional Approach.* Paper presented at the Proceedings of the 35th SIGCSE technical symposium on Computer science education, Norfolk, VN, USA.

Papert, S. (1990). *Constructionist learning*. Boston: MIT Laboratory.

Pausch, R., & Conway, M. (2000). *Alice: Lessons Learned from Building a 3D System for Novices.* Paper presented at the Conference on Human Factors in Computing Systems, The Hague, The Netherlands.

Perkins, D. N. (1993). Person-plus: A distributed view of thinking and learning. In G. Salomon (Ed.), *Distributed cognitions: Psychological and educational considerations* (pp. 88-110). Cambridge: Cambridge University Press.

Rieber, L. P. (2005). Multimedia Learning in Games, Simulations, and Microworlds. In R. E. Mayer (Ed.), *The Cambridge Handbook of Multimedia Learning* (pp. 549-567): Cambridge University Press

Robertson, J., & Good, J. (2004). *Interaction Design And Children* Paper presented at the Proceeding of the 2004 conference on Interaction design and children: building a community Maryland.

Rouse, R. (2004). *Game Design: Theory & Practice*. Plano, TX, USA Wordware Publishing Inc.

Salvage, J. (2001). *C++ coach: Essentials for introductory programming*: Addison-Wesley.

Stroustrap, B. (2004). *The C++ programming language*. Retrieved March 4, 2004, from http://www.research.att.com/~bs/C++.html

Trochim, W. (2001). *The Research Methods Knowledge Base*: Atomic Dog Publishing.

Tucci, L. (2005). *College students continue to shun computer science*. Retrieved March 20, 2006, from http://searchcio.techtarget.com/originalContent/0,289142,sid19_gci1096260,00.html

Vegso, J. (2005). Interest in CS as a major drops among incoming freshmen. *Computing Research News, 17*(3), 236-248.

Watzman, A., & Spice, B. (2006). *Carnegie Mellon collaborates with EA to revolutionize and reinvigorate computer science education in the US*. Retrieved March 22, 2006, from http://www.alice.org/index2.html

White, B. Y., & Frederiksen, J. R. (2000). Technological tools and instructional approaches for making scientific inquiry accessible to all. In M. J. Jacobson & R. B. Kozma (Eds.), *Learning the sciences of the 21st century: Research, design, and implementing advanced technology learning environments* (pp. 321-359). Hillsdale, NJ: Lawrence Erlbaum Associates.

Direct reprint requests to:

Dr. Edward R. Sykes
Professor, Computer Science
School of Applied Computing and Engineering Sciences
Sheridan Institute of Technology and Advanced Learning
1430 Trafalgar Road, Oakville, Ontario, Canada, L6H 2L1
e-mail: ed.sykes@sheridanc.on.ca