Mediated Transfer: Alice 3 to Java

Wanda Dann
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
1-412-268-9959
wpdann@andrew.cmu.edu

Dennis Cosgrove
Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
1-412-268-4074
dennisc@cs.cmu.edu

Don Slater
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
1-412-268-4370
dslater@cs.cmu.edu

Dave Culyba
Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
1-412-268-4074
dculyba@andrew.cmu.edu

ABSTRACT
In this paper, we describe a pedagogy for an undergraduate programming course using Alice 3 and Java. We applied the educational theory of mediated transfer to develop a new version of the Alice system and accompanying instructional materials. The pedagogy was implemented and tested over two years. Student test scores in experimental course sections showed a dramatic jump of at least one letter grade over test scores in more traditional sections of the same course.

Categories and Subject Descriptors
K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms
Measurement, Experimentation, Human Factors

Keywords
Alice, Java, Mediated Transfer

1. INTRODUCTION
In this project, we developed and tested a pedagogy (teaching techniques and instructional materials) for a college level first-year programming course that uses Alice 3 and Java. Alice takes advantage of the high level of interest and motivation students find in video games and animated films. Since its release in 2004, Alice 2 and other highly innovative visualization tools have been increasingly adopted in K-12 schools as an educational tool for introducing computational thinking and fundamental programming concepts. We want to build on this framework and encourage students to continue learning computing in more advanced courses (AP CS and CS1 courses). (CS1 is defined here as a typical undergraduate, first course in programming.) Java is a widely used language of instruction in first-year college programming courses and is the language choice in the current AP Computer Science course.

In terms of technology, we built Alice 3 and a custom plugin for a Java integrated development environment (IDE) that translates the Alice 3 abstract syntax tree (AST) into Java code. In terms of pedagogy, we developed a tailored set of instructional materials that integrates the Alice problem solving strategy (originally introduced using Alice 2 [5]) with support for mediated transfer teaching techniques known as "bridging" and "hugging." This Alice 3 to Java approach is designed to enable students and instructors to transfer concepts learned in the context of Alice animations to programming using a production-level language.

We conducted a two year study of the effectiveness of this Alice 3 to Java approach in terms of student achievement, as compared to a course using only Java. Our hypothesis was that the Alice 3 to Java approach would maintain the strengths of the approach originally introduced with Alice 2 and also manage a transfer of concepts to learn programming in a production level language in a CS1 course with standard curriculum concepts [1].

2. EDUCATIONAL TOOLS
Multiple tools are available for introducing Java programming at the high school and undergraduate college level. In this section, we describe Alice and two well-known tools for teaching/learning Java. The purpose is to clarify the distinctions between Alice 3 and currently available tools for teaching Java that have some visual component.

2.1 Alice
Alice is a programming environment specifically designed as a teaching/learning tool to enable novice programmers to create animations and games using 3D worlds[16, 17]. In Alice, hundreds of 3D models (e.g., people, animals and props) are

Conference'11, Month 1–2, 2010, City, State, Country.

provided in a gallery of Java classes. Instances of these classes are created to populate a virtual world. The Alice IDE provides a drag-and-drop editor (not text-entry) to create program code that animates these objects, thereby eliminating syntax errors by the novice programmer. Alice was designed by studying how novices try to describe the motions of objects in a 3D world, and then modifying Alice to reflect these observed expectations [18]. Alice makes use of program visualization and enables students to immediately see how their animation programs run, enabling students to easily understand the relationship between each individual programming statement and construct and the corresponding behavior of objects in their animation.

2.2 BlueJ

BlueJ [20] is an educational tool used in teaching with an objects-first methodology. BlueJ is a text-entry Java IDE in which the user generally starts with a predefined set of classes. The class structure is presented graphically, in UML-like fashion. The user can create objects and invoke methods on these objects to illustrate their behavior. Later, users may create their own classes. (BlueJ provides a graphical representation of these classes as well.)

2.3 Greenfoot

Greenfoot[12] is an interactive environment that enables students to develop 2D graphic applications such as simulations and games. Greenfoot works with a text-entry Java IDE. At its most fundamental level, Greenfoot has two built-in classes, *World* and *Actor*. The user creates a program in Greenfoot by declaring sub-classes of Actor and then implementing an *act* method, in standard Java, for each sub-class. Execution of a Greenfoot program consists of a built-in main loop that iteratively calls each actor's *act* method.

3. PREVIOUS WORK

Alice 2 has been used successfully as an intervention to draw at-risk students (who are disproportionately female or underrepresented minorities) into computing [4,5,6]. At-risk students were defined as those students who had demonstrated less success in math and/or those who had little previous programming experience. A textbook [5] was developed and pilot tested in introductory computer programming courses (pre-CS1) offered at Saint Joseph's University and Ithaca College. Additionally, a detailed set of curricular materials [8], including several different curricular models (with complete lecture notes), laboratory exercises, solutions, exams, assignments, sample student projects, and other material, was created.

The primary results of this investigation [14] were:

- The average grade for at-risk students exposed to Alice was a 3.0 GPA in CS1, which is comparable to the grades of students who were at no risk or low risk. The average grade for at-risk students not exposed to Alice was a 1.2 GPA in CS1.
- 88% of at-risk students exposed to Alice enrolled in CS2 after CS1. Only 47% who were not exposed to Alice enrolled in CS2. ($p < .05$, chi-squared)

Investigators in this study concluded that the Alice approach doubled retention rates of at-risk students and increased achievement by at least one letter grade.

Other investigators have performed studies in various courses and reached supportive conclusions. For example, Mullins, Whitfield, and Conlon[15] concluded: “Retention data shows that the incorporation of Alice into the programming sequence has increased the number of students that pass the courses and decreased the number of withdrawals. Also, the number of students enrolling in Alice has increased by 10% as the number of majors has dropped by 50%.”

Instructional materials have been designed for and tested with pre-CS1 and pre-AP courses. Over the last six years, dozens of Alice professional development workshops have been conducted in which the curricular materials and the Alice approach for pre-CS1 and pre-AP were presented.

In National Science Foundation-sponsored workshops[9], surveys were used to obtain feedback from participants regarding their intended target audience and their plans for using Alice 2 in their courses. One clear survey result was a significant demand for curriculum and instructional materials that could be used to blend the Alice approach with Java in a "regular CS1" course. The primary reason for this demand is the reality of reduced budgets and limited resources in collegiate Computer Science departments, which limits their ability to offer a pre-CS1 course. Community colleges face similar problems and are further burdened by the constraints of a 2-year curriculum and the need to adhere to articulation agreements.

To respond to this demand, a major question had to be answered: How can Alice be used to teach/learn fundamental programming concepts in an engaging context and then apply those same concepts in a text-based Java environment? Early attempts to start with Alice 2 and then move to a professional programming language in the same course demonstrated that "new techniques are needed to improve student confidence during the transition from Alice’s graphical, syntax free, storytelling environment to object-oriented textual programming."[19]

Three textbooks have been published in an effort to address this issue [2, 3, 13]. In each of these texts the authors (including the PI on this study) use Alice 2 examples to introduce a programming concept and then a traditional Java program that uses the same concept. What is missing is the ability to transfer the Alice animation directly into Java code.

4. MEDIATED TRANSFER

The goal of an Alice 3 to Java approach is to take advantage of Alice in developing an intuitive understanding of both object-oriented and fundamental programming concepts and then directly transfer the Alice program directly into Java and programming in a text-based IDE. By using the exact same example in both Alice and Java, we can mediate a transfer of concept.

4.1 Educational Theory

Educational theorists are adamant that learning should be transferable. [7, 11, 21] That is, what is learned in one context should be employable in another context. Although we may expect transfer of learning, it does not necessarily occur "on its own."

A good illustration is the following anecdote[21]: A Physics professor presented and solved this problem in lecture: "A ball weighing 5 kg is dropped from the top of a building that is 100 meters high. How many seconds later does the ball hit the ground." Then, this problem was on the next exam: "A hole in the ground is 150 meters deep. A ball weighing 7 kg is dropped into the hole. How long does it take for the ball to reach the bottom of

the hole?" One student complained to the professor that the problem was unfair because "We didn't have any hole problems!"

While we might dismiss this as just a funny story to tell our colleagues, it is actually a common student experience in any problem-solving discipline. In our experience, it is not unusual to observe students who are bewildered and not at all sure of how to begin designing and implementing program code for an assigned project, even though a similar problem was previously demonstrated in lecture/presentation.

4.2 Teaching for Transfer

The art of teaching for transfer, known as "mediating transfer" is an active research field [7, 11, 21]. Perkins (Harvard Graduate School of Education) and Salomon (University of Arizona) [21] define two broad categories of techniques that teach for transfer. The first category is "bridging," in which the teacher helps students build a bridge from the context in which a concept was learned into other potential contexts. Bridging is in the form of meanings, generalizations, and insights. The second category is "hugging," in which the teacher makes the learning situation more like the situations in which transfer is expected.

In this Alice to Java approach, we apply a "bridging technique" of using analogies, and the reasons for them, to encourage students to abstract the concept and recognize other contexts in which it may be applied. For example, in Alice we introduce the concept of parameter data types by writing a procedure to have a dragon fly toward a target object, a given distance, in a given amount of time. The target object's data type is Person and the distance and amount of time are each of type Double. We then discuss the example with students, using the analogy of ordering a pizza on their cell phone. The waiter taking an order will ask for the size of the pizza (Double), whether or not it is to be delivered (Boolean), and so forth. Each of these items is a different data type. Then, we ask students to work in pairs to brainstorm other situations where information must be supplied in order to carry out some action. The idea is to integrate into Alice to Java instructional materials the use of techniques that teach for transfer. Using analogies and encouraging students to develop their own generalization of a concept in different contexts is just one example of a bridging technique.

Fortunately, using bridging techniques in instructional materials is software independent. And, many teachers already use these techniques in their teaching styles. However, using a "hugging technique" for transitioning from Alice to Java implies that writing a program in Alice needs to be more like writing a program in Java. Clearly, this means the Alice software must undergo changes – but at what cost? Alice allows students to assemble programs using a drag-and-drop editor. An advantage of the drag-and-drop editor is that students can focus their attention on understanding a fundamental programming construct without the initial distraction and frustration of syntax details.

To maintain a drag-and-drop IDE and also "hug" Java, Alice 3 has been designed to provide a set of preference options that allows the student to view Java code with greater syntax details than in Alice 2. (Note: Alice 2 had a similar option, but the code was "Java-like," whereas Alice 3's Java display is a far more accurate representation.) Figure 1 illustrates the preference options for Java code in Alice 3. Figure 2 illustrates the Java code, with all parentheses, quotes, commas, and semicolon syntax detail.

We found that the more accurate representation of Java code in Alice 3 is a step in the right direction, but it is not sufficient to truly "hug" a Java text-editor environment. For more effective transfer, we developed a plugin for NetBeans (an open source Java IDE[10]) that allows students to transfer their Alice project directly into Java, as illustrated in Figure 3.

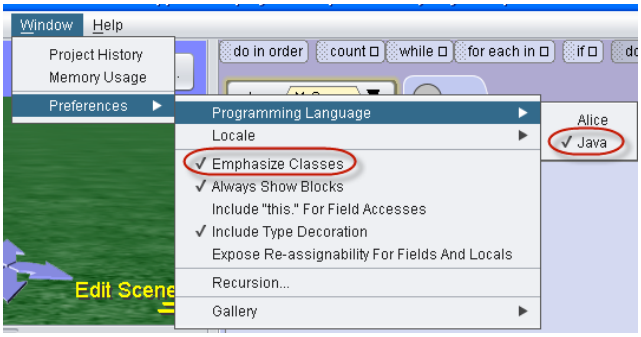

Figure 1. Java options in Alice 3

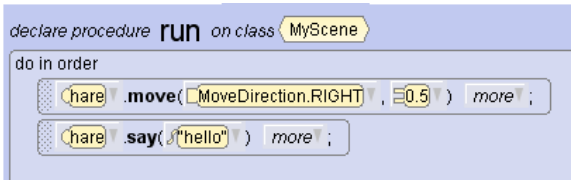

Figure 2. Java code in Alice 3

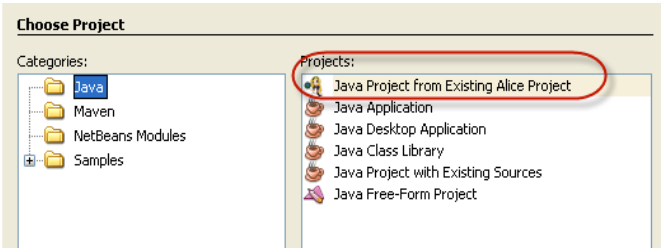

Figure 3. Transferring an Alice Project to Java text-based IDE

When a student transfers their own Alice project from Alice 3 to a Java IDE, the context does not change. They are still working with the same animation program as was created in Alice but now the code can be modified using traditional text editing. Figures 4 and 5, below, show an example of student code for the following assignment:

> *Build a commercial for Bug's Gym, where a frog catches a bug and gets a surprise when the bug grabs the frog's tongue, lifts the frog off the ground, and carries it away to drop in a nearby pond. The commercial message is "Learn how to handle bullies at Bug's Gym."*

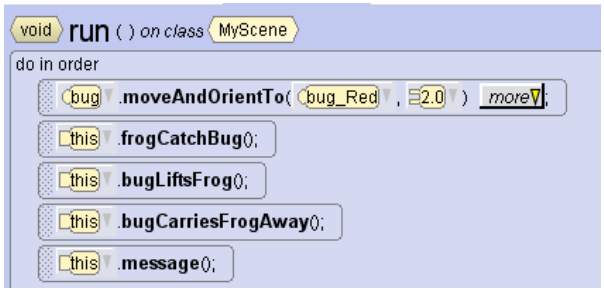

Figure 4. Alice 3 code for Bug's Gym

```
public void run() {
      bug.moveAndOrientTo( bug_Red, 2.0);
      this.frogCatchBug( bug);
      this.bugLiftsFrog();
      this.bugCarriesFrogAway();
      this.message();
}
```

Figure 5. Alice 3 code transferred to Java IDE

With each transfer of code from Alice 3 to a Java IDE, the instructor uses "hugging" techniques to directly connect Alice code to Java code. For example, both Alice 3 and Java IDE are displayed on a projector and lines are literally drawn from Alice statements to equivalent Java statements. Then, students are given a previously unseen Alice code segment and asked to write (on paper) the equivalent code in Java. In this same lab exercise, students are also asked to modify the code in the Java IDE to add a second frog and, at runtime, ask the user to select which frog that will perform the actions with the bug. This modification is done in the Java IDE with text-entry and requires that some methods be modified to use parameters.

5. TESTING AND EVALUATION

5.1 Historical Data

For comparison purposes, historical data was collected from a final exam originally administered in the Carnegie Mellon University (CMU) 15-100 course sections in the Spring semester, 2006. At that time the CMU 15-100 course was a course in Java programming and students in the course included both majors and non-majors. Course instruction consisted of three 50-minute lectures/presentations and one 50 minute recitation/lab session per week. Alice 2 was introduced during the first two weeks of the course, before switching over to Java for the remainder of the course. Mediated transfer techniques were not used.

Since final exams and summaries are kept on file for at least 3 years, access to a historical record of raw scores, broken into the topic sections of the exam (Parts 1–6) as well as final scores (total 100 possible points) were available. Student identities were protected as no student names were included in the summary.

The content of questions in each part is listed here:

Part 1: Expression evaluation (arithmetic, Boolean, String)
Part 2: Control Structures (conditionals, iteration)
Part 3: Arrays of Primitives
Part 4: Work with a class definition
Part 5: Work with arrays of objects

Table 1 summarizes the historical data from that exam, taken by students in one of the investigator's class sections. The number of students tested was 67 (N = 67).

Table 1. Historical Achievement Data, Spring 2006

Spring 06	Part 1	Part 2	Part 3	Part 4	Part 5	Part 6	Total
Possible Points	10	15	15	20	20	20	100
Average	7.45	6.84	9.42	14.7	9.21	13.2	60.8
Percent	74	46	63	74	46	66	

5.2 First Trial

In the Fall semester of 2009, experimental sections of CMU 15-101 were offered. The course instructors used the Alice 3 to Java mediated transfer approach and curricular materials. Course instruction time for 15-101 was the same as the previous 15-100 course consisting of three 50-minute "lectures/presentations" and one 50-minute "recitation/lab" per week. One of the participating instructors had also taught the 15-100 course in 2006 and was familiar with the standard course content.

At the conclusion of the Fall 2009 semester, students in 15-101 took the same exam (with the modification that topics in Parts 5 & 6 were merged to use a 3D animation example as a basis for the questions). The number of students tested in Fall 2009 was 50, most of whom were non-majors and had little if any prior programming experience (N = 50). The data shown in Table 2 summarizes results in the Fall 2009 semester.

Table 2. Experimental Achievement Data, Fall 2009

Fall 09	Part 1	Part 2	Part 3	Part 4	Parts 5 & 6	Total
Possible Points	15	20	20	25	20	100
Average	13.06	17.46	16.3	21.26	16.88	84.96
Percent	87	87	81	85	84	

Table 2 indicates that Parts 5 & 6 were merged. As described above, Parts 5 & 6 of the original exam (Spring 2006) were blended into Part 5, using a programming example for an Alice 3 animation. Although the program is for an animation, it was written on paper in Java. Students were asked to write their code in Java (as on the original exam). Even with these disclaimers, we are uncomfortable making any claims with regard to Parts 5 and 6 of the exam. For the purpose of full disclosure, the tables include all data. For the purpose of comparison, however, the figures below will display only Parts 1 – 4.

The best (most scrupulous) evaluation of this data is to compare percentage scores where Part 1 is compared to Part 1, Part 2 to Part 2, and so forth. Comparisons of respective parts of the exam show a consistent gain of at least 10 % (one letter grade) and the gain in Part 2 is nearly double, as can be seen Figure 5. An

ANOVA analysis of the data yielded $p < 0.001$ for each Part of the exam as well as for the total scores.

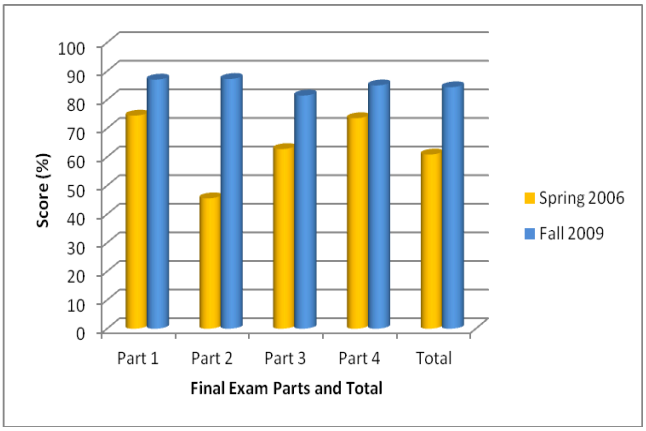

Figure 5. Comparison of Parts, Spring 2006 and Fall 2009

5.3 Second Trial

The data collected and analyzed in Fall 2009 showed a clear and dramatic increase in scores in the respective parts of the exam. Being wary about making broad claims based on a single trial run of the instructional materials and curriculum, we made a decision to extend the study and run a second trial.

In the Fall 2010 semester, an experimental section of CMU's 15-101 (Alice to Java) course was conducted once again, using instructional materials developed in this project. The number of students involved in this iteration was 28 (n = 28). As in the first trial (Fall 2009), the students were primarily non-majors and had little or no previous programming experience. To control as many other factors as possible, the same instructors taught the courses and used the same instructional approach. Further, course instruction time for 15-101 remained consistent, three 50-minute instruction sessions and one 50-minute recitation/lab sessions per week.

The evaluative exam was administered for this test group and raw scores and percentages for this iteration are listed in Table 3. Again, all raw data is reported, for purposes of full disclosure. An ANOVA analysis of the data yielded $p < 0.001$ for each Part of the exam as well as for the total scores.

Table 3. Second Trial, Fall 2010

Fall 10	Part 1	Part 2	Part 3	Part 4	Parts 5 & 6	Total
Possible Points	15	20	20	25	20	100
Average	13.1	16.0	16.5	21.75	14.54	81.52
Percent	87	80	83	87	73	

5.4 Summative Comparison

The exam score data can be interpreted as a guide to the achievement of students in the course. In each semester, the final exam was written in "pure Java." That is, all questions were stated and all answers were written using Java.

Two trials were run to obtain reliable data for comparison, one in Fall of 2009 and one in Fall of 2010. The original exam (Spring 2006) was taken by a diverse group of students, including both majors and non-majors (though the majority were non-majors). The exam taken in Fall 2009 and in Fall 2010 was taken almost entirely by non-majors. Figure 7 summarizes the results of the two trial runs with the historical data.

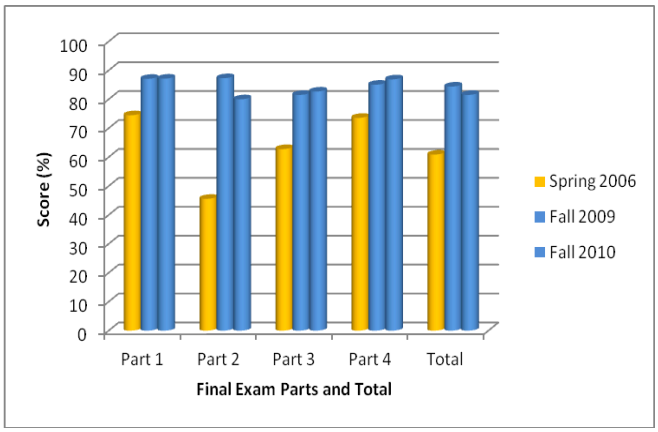

Figure 7. Summative comparison

In the summative comparison for Parts 1 – 4, average student scores (expressed in percentage) in Fall 2009 and 2010 experimental sections were consistent. This provides evidence that the test results are reliable. In viewing the overall scores, student achievement in the experimental sections averaged at least one letter grade above those in the more traditional Java course. This confirms the hypothesis that the Alice to Java approach would maintain the strengths of the approach originally introduced with Alice 2, in terms of student achievement. (Recall, similar results in the Alice 2 study conducted by [14].)

Of particular interest is the large increase in scores in Part 2 of the exam. The focus of questions in Part 2 is Control Structures (conditionals and iteration). We do not have evidence as to the specific cause of this dramatic jump in scores in this part of the exam. This suggests a topic for future study.

In summary, data collection and analysis indicates that using Alice 3 to introduce fundamental concepts combined with materials designed to teach for transfer from Alice 3 to Java has a statistically significant positive impact on students' learning.

5.5 Other findings and future work

Although this study was focused on evaluating the effectiveness of an Alice 3 to Java approach using mediated transfer, other findings are noteworthy.

One finding is that for transfer of concept, inheritance needs to be more fully implemented in Alice 3 than it was in Alice 2. The lack of fully implemented inheritance constrains the ability to illustrate inheritance concepts in Alice and then mediate a transfer of those concepts to Java.

A second finding is that transferring Alice code containing concurrency and events to Java results in complex code that forces the introduction of inner classes early in the curriculum. But, inner classes are not a common topic in a CS1 course.

These two findings are catalyzing revisions in Alice 3 and will form the basis of future work.

6. ACKNOWLEDGMENTS

This work is based on upon work supported, in part, by the National Science Foundation under grants: DUE #1007631 and DUE #0903271.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] ACM Joint Task Force for Computing Curricula. 2005. Computing Curricula 2005. Retrieved September 2, 2011 from http://www.acm.org/education/education/curric_vols/

[2] Adams, J. 2007. *Alice in Action with Java*. Thompson Course Technology: Boston, MA.

[3] Dann, W., Cooper, S., and Ericson, B. 2009. *Exploring Wonderland: Java Programming Using Alice and Media Computation.* Prentice-Hall: Upper Saddle River, NJ.

[4] Cooper, S., Dann, W., & Pausch, R. 2003. Teaching objects-first in introductory computer science. *Proceedings of the 34th SIGCSE technical symposium on Computer science education* (Reno, Nevada, USA).ACM Press.

[5] Dann, W. P., Cooper, S., & Pausch, R. 2005. *Learning to program with Alice*. Prentice-Hall: Upper Saddle River, NJ.

[6] Dann, W., Dragon, T., Cooper, S., Dietzler, K., Ryan, K., & Pausch, R. 2003. Objects: Visualization of behavior and state. Proceedings of the 8th annual conference on innovation and technology in computer science education, Thessaloniki, Greece, 84-88.

[7] Forgarty, R.; Perkins, D.; & Barell, J. 1991. *The Mindful School: How to Teach for Transfer.* Palatine, IL: IRI/Skylight Publishing.

[8] Alice 2 instructional materials . Retrieved September 2, 2011 from http://www.aliceprogramming.net

[9] Alice workshops – 2011. Retrieved September 2, 2011 from http://www.aliceprogramming.net/workshop2011.html

[10] NetBeans download. Retrieved September 2, 2011 from http://netbeans.org/downloads/

[11] Ip, Alex. Transfer of Learning. Retrieved September 2, 2011 from http://www.cdtl.nus.edu.sg/ideas/iot18.htm.

[12] Kolling, M. 2009. *Introduction to Programming with Greenfoot:*. Prentice-Hall: Upper Saddle River, NJ.

[13] Lewis, J. and DePasquale. 2008. *Programming with Alice and Java*. Addison Wesley: Boston, MA.

[14] Moskal, B., Lurie, D., & Cooper, S. 2004. Evaluating the effectiveness of a new instructional approach. *Proceedings of the 35th SIGCSE technical symposium on Computer Science Education. (*Norfolk, Virginia).

[15] Mullins, P., Whitfield, D., and Conlon, M. 2008. Using Alice 2.0 as a first language. *Journal of Computer Science in Colleges,* 24(3), 136-143.

[16] Pausch, R., & Forlines, C. 2000. Alice: Model, paint & animate — easy-to-use interactive graphics for the web. *SIGGRAPH Comput. Graph., 34*(2), 42-43.

[17] Pierce, J., Cobb, T., & Pausch, R. 1998. Alice. *ACM SIGGRAPH 98 Conference abstracts and applications*. (Orlando, Florida, United States).

[18] Pierce, J. S., Christiansen, K., Cosgrove, D., Conway, M., Moskowitz, D., Stearns, B., et al.1998. Alice: Easy to learn interactive 3d graphics, *CHI 98 conference summary on* Human factors in computing systems. (Los Angeles, California, USA).

[19] Powers, K., Ecott, S. and Hirshfield, L. 2007. Through the looking glass: teaching CS0 with Alice. *Proceedings of the 38th SIGCSE technical symposium on Computer Science Education.* (Covington, Kentucky, USA).

[20] Rolling, M. & Rosenberg, J., Guidelines for teaching object orientation with Java. 2001. In *Proceedings of the 6'h annual conference on Innovation and Technology in Computer Science Education* (Canterbury, England, June, 2001), 33-36.

[21] Salomon, G., & Perkins, D. 1988, September. Teaching for transfer. *Educational Leadership*, 22-32.