Initial Results of Using an Intelligent Tutoring System with Alice

Stephen Cooper
Computer Science Department
Stanford University
Stanford, CA 94305
1.650.723.9798
coopers@stanford.edu

Yoon Jae Nam
Computer Science Department
Stanford University
Stanford, CA 94305
1.650.455.0064
namy@stanford.edu

Luo Si
Computer Science Department
Purdue University
West Lafayette, IN, 47906
1.765.496.9381
lsi@cs.purdue.edu

ABSTRACT
This paper describes the initial steps taken towards incorporating an intelligent tutoring system (ITS) into Alice. After initially describing an ITS, the paper focuses on the development of several tutorials for teaching specific introductory programming concepts that have been created using stencils. Initial results concerning usability and effectiveness of these stencil-based tutorials are provided.

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

General Terms
Algorithms, Measurement.

Keywords
Intelligent tutoring system, Alice, tutorial.

1. INTRODUCTION
Computing is becoming popular again! From the Khan Academy [27] finally offering instruction on how to program, to Jan Cuny's ambitious plans to recruit 10,000 high school teachers to provide computing capacity at the K-12 level [6], to the nearly 200,000 people who signed up for Thrun and Norvig's online Artificial Intelligence course [18], there has been an increased interest in learning computing. As Mark Guzdial noted in his keynote address at ITiCSE in July 2011, it has become increasingly difficult to use the same apprentice-based approach that has most commonly been used to teach introductory computer science (that has typically focused on teaching programming) over the past 40 or so years. It is too expensive. It doesn't scale well. And it depends quite heavily on the quality of the tutor/master.

This work proposes to use an Intelligent Tutoring System (ITS) to augment or possibly replace the role of the instructor. It is incorporated into Alice.

ITiCSE'12, July 3–5, 2012, Haifa, Israel.

Intelligent Tutoring System(s) (ITS): An ITS is a computer system that provides customized instruction or feedback to students without much intervention from human teachers. An ITS can bring several benefits for delivering effective instruction: (1) An ITS can provide individualized instruction that allows all students to access the same curriculum with different entry points and learning tasks that are tailored to students' needs; (2) An ITS can enable students to achieve similar proficiency levels in a more efficient way; and (3) With appropriate design, an ITS can enable human teachers to focus on a small subset of students who need extra help besides the assistance provided by the ITS instruction, and thus delivering more effective instruction.

Alice: Alice is a programming environment designed to enable novice programmers to create 3-D virtual worlds, including animations and games [19, 20] as they learn to program. Students use a drag and drop editor to manipulate the movement and activities of these objects. Alice was designed through an iterative process of studying how novices try to describe the motions of objects in a 3-D world, and then modifying Alice so that the novices' expectations would be met [21]. Alice makes use of program visualization to allow students to immediately see how their animation programs run, enabling students to easily understand the relationship between the programming statements and constructs and the behavior of their animations. In Alice, students learn the basics of computing, but where the objects of concern are actors and scenes in a virtual world. Alice programs have a strong object-oriented flavor, allowing students to control the appearance and motion of objects, have objects respond to mouse and keyboard input, or do any sort of computation that would normally be done in an introductory programming class. Alice was chosen as the environment into which to place the ITS due to its popularity as an introductory programming language/environment.

The intent of the ITS is twofold: Initially, it is to be used to detect student progression through the tutorials. When does a student appear to have mastered a particular programming concept, and is ready to move on to the next concept? The second intent for the ITS is to detect when a student appears to be struggling while programming in Alice. Based upon where the ITS detects the student to be struggling, it will suggest one or more tutorials for the student to attempt to complete, prior to returning to the student's original program.

This paper's focus is on the tutorials that have been created to support the ITS. While a brief description of ITS's is provided, the intent is to do no more than provide a justification for our

research approach. Prior to covering the overall system, it is necessary to argue that the tutorials "work" in the sense of helping students to understand a particular programming concept.

2. RELATED RESEARCH

ITS: The earliest ITSs date back at least to the 1970s. They have increased in popularity since the 1990s as more computers have been available in education applications and more intelligent methods have been utilized to improve the instructional effectiveness of ITSs. Intelligent tutoring systems have been utilized in different disciplines including mathematics, physics, chemistry and biomedical education. Some successful examples of ITSs include: ANDES for physics [8], AutoTutor for Newtonian physics, computer literacy, and scientific reasoning [11], PACT for algebra [16], SHERLOCK for electronics troubleshooting [17], and a tutor system for Chemical Engineering [4]. For example, the PCAT cognitive tutor for algebra has been studied in classrooms with 470 students for one whole year. Students in the experimental classes have been shown to outperform students in control classes (i.e., standard classes) by 50-100% on targeted practical problem solving skills and by 10-25% on standardized tests.

Some prior research work has been conducted to develop tutoring systems for computer programming education. Early systems tended to focus on identification of student errors in completed programs. These systems include Proust and Chiron [22], MENO II [26], and LispTutor [2]. Some tutoring systems use enhanced integrated development environments (IDEs). IDE tutoring systems (e.g., [23]) generally help students to identify syntactic or logic errors and students need to figure out how to fix these errors based on the information provided by the systems. The cognitive tutors go beyond the IDE approach by providing more direct feedback/suggestions. Cognitive tutors are based on the Adaptive Control of Thought-Relational (ACT-R) theory in cognitive psychology [1, 3, and 5]. In particular, cognitive tutors analyze both domain knowledge and problem-solving strategies. A cognitive model is built to represent the paths that students may use to solve problems. When a student works on the problem, the cognitive model identifies the solution path of the student and provides corresponding feedback/suggestions with respect to how the student tries to solve the task. The cognitive tutors have been applied to teach some basic programming concepts and production rules in programming [13, 5]. It has been shown by Jin in [13] that CS1 students taught with cognitive intelligent tutors outperform students in standard classrooms by more than 50% in solving targeted programming questions. However, it is generally quite time consuming to construct the cognitive models on many concepts and problems for a large population of students.

The Intelligent Learning Material Delivery Agent (ILMDA) approach uses a case-based reasoning method to suggest instructional materials through analyzing students' performance in answering questions of different topics [24, 25]. In particular, ILMDA stores different cases of instructional strategies and specific solutions. The case that best matches a current instructional situation will be retrieved and adapted in an online manner. A set of real world studies have been conducted by Soh et al. for a large CS1 course in two years to demonstrate that the intelligent ILMDA tutoring system can deliver appropriate exercise problems to students, which help them learn the materials more effectively and efficiently. WebWork is a Web-based homework delivery and grading system, which automatically compiles students' submissions and evaluates on some datasets [7, 10]. However, WebWork does not provide enough feedback/suggestions to facilitate students' learning. Its focus is more towards assessment of student learning/knowledge.

Tutorial systems: The majority of tutorial systems tend to present text based instruction as a separate window with images describing what the user should be doing. Such systems have often been shown to be hard on users [15]. Other approaches have included information within the actual context of the application itself. Such systems have included graphical overlays [9] and embedded videos [12], and have shown limited success. The system chosen here is Kelleher's stencils [14], a graphical overlay system that has been shown to be successful (in terms of speed and retention) in helping students to learn an application.

3. TUTORIALS

3.1 Stencils

Kelleher's stencils provide a means to interactively teach Alice within the Alice environment itself.

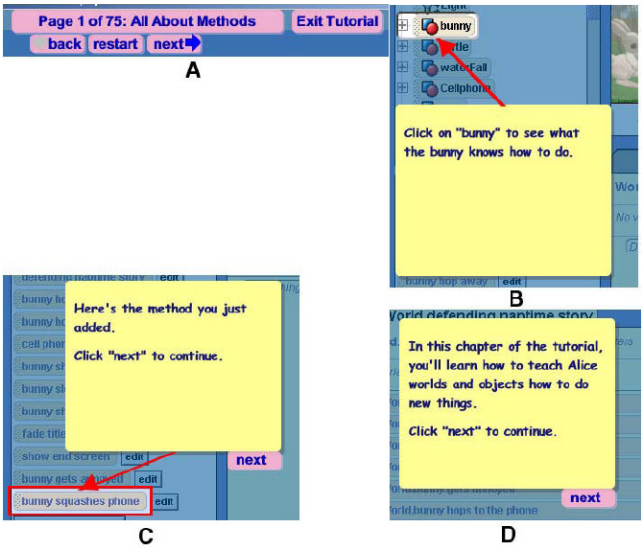

Figure 1. From [14], image A shows the navigation bar, B shows a hole with an associated note, C shows a frame with a note, and D is a standalone note.

Whereas Kelleher used stencils to teach the Alice environment, we have designed stencil tutorials to actually teach programming concepts. The appeal of Kelleher's stencils is that they provide a lovely vehicle to allow students to incrementally develop solutions to a problem. And the units of incremental development can be made arbitrarily large or small.

3.2 Teaching Alice

We developed a total of thirteen tutorials that teach basic programming concepts with which novices often have difficulties, including parameter passing, nesting of control statements, repetition (definite and indefinite loops), and list manipulation. These topics were chosen either because of their centrality in introductory programming or because they were reported as being harder to teach (as far as students' abilities to understand the concepts) as part of the annual educator survey sent out to the 3000+ member Alice educator mailing list. Along with each tutorial, a paired activity was designed to measure the extent to which the student learned the concept presented in the tutorial.

For each tutorial, we created a storyline and stated the goal of the tutorial, in order to make the tutorials engaging. This fits with our general problem solving approach we use to the teaching of Alice.

Table 1. List of tutorials and concepts taught

No.	Tutorial Name	Concepts taught
1	Snowpeople	Nesting of control statements – a general introduction to Alice programming
2	Taxi	While loops – simple conditions
3	Bunny baby	While loops – complex loop condition
4	Penguins	While loops – using a function call within the loop condition
5	Rescue	While loops – multiple loops with user-defined methods being added
6	Animals	If statement and While loops – nesting of these structures
7	Husky	If statement and While loops – a second, more complex nesting of these control structures
8	Mars Rover	For loops and parameter passing – using a simple numeric parameter
9	Jump and Spin	Parameter passing – multiple types, and more complex use of parameters
10	Moo	Lists and parameter passing
11	Gather	Parameter passing with multiple parameters
12	Midpoint	Recursion and While loops
13	Exercise	Parameter passing and lists, including some Alice idiosyncrasies

In the snowpeople tutorial, the student has to create a scene where a snowman unsuccessfully tries to grab a snowwoman's attention. In order to create this scene, the student has to nest control statements inside one another.

The goal of the taxi tutorial is to teach the concept of while loops to the student. The student has to help Alice catch up with a taxi that is already moving forward. The student has to make Alice and the taxi move while Alice has not reached the taxi's rear.

The bunny baby tutorial teaches students how to use a complex while loop condition. Three bunnies are introduced in the tutorial – a mother, a father, and a baby. Initially, the mother and the father are away from the baby. The bunny baby starts to cry, and his parents come back to the baby to soothe him. The student has to use a complex while condition to make the bunny baby cry while any of his parents is not present.

In the penguin tutorial, the student has to use a while loop to switch the positions of two penguins. The exact distance between the penguins is not given, so the student has to use a while loop and a relative distance function to switch the penguins. Initially, the two penguins are apart by a few meters; one penguin is on the other's left. The student has to use a while loop to make the penguins move while they have not switched their positions.

In the rescue tutorial, the student creates a rescue scene where a rescue ship carries people from a spaceship that is crashing into the ocean. This tutorial also makes the student create his/her own methods. The student must first use a while loop to write a method that makes the spaceship fall into the ocean while its body is still above the ocean. Then the student should create two methods, one of which makes the rescue ship approach the spaceship, and the other that makes the rescue ship approach the island. Finally, the student has to use a while loop to put things together and make the rescue ship fly back and forth between the island and the falling spaceship.

In the first part of the animals tutorial, the student has to use an if statement and a while loop to help a chicken cross the street. A bunny is sitting in the chicken's way, so the chicken has to jump over it if it gets close to the bunny. The student is provided with a method that makes the chicken hop forward a certain distance. The student must execute this method if the chicken is close to the bunny. The student must use a while loop to make the chicken move forward while it has not reached the destination at the other side of the road.

The second part of the animals tutorial introduces another way to use an if statement and a while loop to achieve the same goal. This time, the student has to make a hare cross the street by hopping over the bunny. Unlike in the first part of the tutorial, where the chicken moved forward outside the if statement, this time the student makes the hare move forward inside the if statement.

In the husky tutorial, the student creates a scene where a husky chases a chicken. A while loop is used to make the husky chase the chicken while it has not reached the chicken. However, if the husky gets too close to the chicken, the chicken runs away much faster than the husky. As a result, the husky can never catch the chicken. Also in this tutorial, the student learns how to make the camera follow the husky, so that the husky and the chicken do not go out of the student's sight.

In the mars rover activity, the student has to write a method that makes a mars rover draw a square. This method takes a numeric parameter that determines the side length of the square. In addition to learning how to create a method that takes a numeric parameter, the student learns how to use a loop in order to make the rover draw a square.

In the jump and spin tutorial, the student has to write a method that takes an object parameter in order to make several animals jump and spin at the same time.

In the moo tutorial, the student learns how to use a list of objects and a method that takes an object parameter to make five cows moo. The initial goal stated in the beginning of this tutorial is to make the five cows moo at the same time. Then, the tutorial lets the student explore what happens if he/she changes the list tile from "For All Together" to "For All In Order." After the student sees that this change makes the cows moo one by one, the tutorial goes on to show what happens if the student changes the order of items in the list.

In the gather tutorial, the student learns to create a method that takes more than one parameter. There are six animals in the tutorial – two cows, two chickens, and two bunnies. The student has to make the animals gather by species. The student must write a method that takes two object parameters and that makes the animal passed in as the second parameter to go to the animal

passed in as the first parameter. Then the student should use this method to make the animals gather by species.

In the first part of the midpoint tutorial, the student learns to use a while loop in order to help a chicken find the midpoint between two cones. At the beginning of the tutorial, the chicken stands right next to one cone and is facing the other cone. The student must use a while loop to make the chicken move forward while its distance to the first cone is smaller than its distance to the second cone.

In the second part of the midpoint tutorial, the student is introduced to recursion. The student writes a recursive method called "findMid" that makes the chicken find the midpoint of the two cones by first going to the second cone and then tracing back half the distance.

In the exercise tutorial, there are four characters – three classmates and one teacher. In the first part of the exercise tutorial, the student has to make the four characters into a list, and then use that list to make the classmates turn to the teacher. In the second part of the tutorial, the student should use the same list to make the characters raise their arms. Since the student cannot directly access the parts of the item in the list, the student must use a "part named" function in order to make the characters raise their arms.

4. APPROACH

We were greatly influenced by a keynote address John Sweller gave at ICER in 2008. In a conversation with Sweller after the keynote, when asked how cognitive load theory could be applied to the teaching of introductory programming, Sweller proposed providing a great number of examples illustrating each particular programming concept. In the spirit of this suggestion, we chose to build each tutorial as a series of missteps. In other words, we intentionally have the students make all of the common mistakes we have observed as introductory computer science instructors, and observe what happens as a result. Because Alice provides such a strong visual environment, the student immediately sees what is going wrong when the animation "misbehaves". Our intent was to provide students with a collection of relationships between errors and the results of those errors. So, rather than, as Sweller suggests, providing students with a collection of working examples, we instead provide the student with an awareness of what errors lead to what incorrect results. So that later, when the student comes across a particular error, that student will recognize the error and understand what logical flaw led to its creation.

Although the tutorials themselves prevent the students from going on to the next step if the students do not follow the instructions carefully, the tutorials do have the students make mistakes so that the students can learn from them. In almost every tutorial, we introduce many errors that students often make. By having the students go through a mistake and see what happens for themselves, we made our tutorials educational and interactive. In their present form, the tutorials ask students what went wrong, but do not collect student responses, and instead provide our responses in a later tutorial frame.

We illustrate by example, the nature of the errors we introduce with while loops. We have found that while loops are a rich source of student errors, both because the Boolean condition determining whether or not the while loop is to be entered is tricky, as well as from the fact that it is often unclear exactly which instructions belong within the while loop, rather than before or after it.

Mistakes with while loops we introduce in the tutorials include:

- The opposite of the Boolean condition to determine if the while loop should be entered
- Boolean conditions that are off by one (leading to the while loop being entered one too many or one too few times.
- Using a Boolean condition of (A and B) when (A or B) is warranted, and vice versa.
- Errors with negation of Boolean conditions (such as incorrectly trying to apply DeMorgan's laws).
- Having the student place a method that should be called once, prior to entry to the while loop to instead be called within the body of the while loop.
- Having a student place a method that should be called within the body of the while loop to instead be called prior to the while loop or after the while loop.
- Errors involving nesting of control structures (multiple while loops or while loops and if statements).

After each error is made, and the student tries to run the world with the error, the tutorial asks the student to reflect on what went wrong, and how the student knew what was going wrong. As mentioned earlier, we are not presently capturing the student responses – we are simply asking the student to consider what went wrong before providing our answer.

By allowing the students to experience errors before being shown (and building) a correct solution, we try to help the students understand common mistakes.

5. DISCUSSION/RESULTS

We tested our tutorials and activities on nine undergraduates with little or no prior knowledge in computer science, students who self-reported minimal to no prior programming experience. All of the students were asked to complete the initial tutorial with no help. The initial tutorial introduced the students to programming in Alice. After completing the initial tutorial, we had the students complete differing sequences of the other tutorials. Each tutorial topic was designed to be as focused on a single programming concept as was possible. In order to determine that the student had mastered the particular programming topic, we asked the student to solve an alternate problem (we use the term activity, to differentiate it from the act of completing the tutorial). The solution to the alternate problem required an understanding of the topic that was taught in the tutorial. For the activity, we showed the student a youtube video of what a successful solution could look like, and then gave the student an initially setup world, without any code. The student was then given time to complete the activity, and we determined the extent to which the student's solution actually solved the problem posed in the activity. In addition to observing the student working on the activity, we modified Alice to capture all of the student's drags and drops, as well as the student's intended drops (when an attempt to drag and drop a syntactically invalid tile would be blocked by Alice). On average, each student spent 20 minutes on a tutorial and 11

minutes on an activity. The initial Alice tutorial took slightly longer to complete than the others.

Students were asked to describe their reactions to the tutorials. In 68% of the trials, the participant answered that the instructions in the tutorial were very clear, with the remaining students reporting that the instructions were mostly clear. In 73% of the trials, the participant answered that they learned a lot from the tutorial, with the remainder reporting that they had learned a medium amount from the tutorials. More significantly, all participants were able to correctly solve the problem posed by the activity (associated with the tutorial). Somewhat surprisingly, there were a couple of cases where we gave the student an activity that was harder than the associated tutorial (an activity that was paired with a more advanced tutorial). In each of these cases, the student was still able to complete the activity (though it took a fair bit more time).

In 60% of the trials, the participant answered that the tutorial was easy to walk through. In 30% of the trials, the participant answered that the tutorial was of medium difficulty. Based on the first students' observations, changes were made to the tutorials so that later students indicated increasing ease with the tutorials.

We tested the tutorials on both Mac and Windows. There was no significant difference between students who used Mac and those who used Windows. We tested the tutorials on students with different majors, spanning Social Science, Natural Science, Engineering, and Arts and Humanities. There was no significant difference between the performances of the students with different majors. However, some Social Science and Arts and Humanities majors performed unusually well, possibly because they seemed to have read and understood the tutorial instructions better than others.

The tutorial that took the longest time to complete was the snowpeople tutorial, which we gave to the students for their first tutorial experience. It took some time for the students to get used to the interface and the tutorial system. After the first tutorial, the students finished the tutorials in much less time and with fewer errors.

We found two somewhat significant problems with the use of Kelleher's stencils. The first was that each step of the tutorial needed to be completed exactly correctly by the student. While this makes sense in general, Alice includes a Do Together structure where the instructions within the Do Together block all occur simultaneously. Thus, the order of instructions should not matter. However, the stencils did require that the students' instructions exactly matched the "canonical solution", which seemed wrong from a pedagogic perspective. While we explained this situation within the stencil notes, this is clearly "wrong" with respect to how the stencils should work with respect to the tutorials. In the same vein, students who added extraneous parameters on method calls (such as the Style parameter), parameters that really have nothing to do with the lesson the tutorial was trying to teach, got their code flagged as being incorrect by the stencil system.

The second problem with the stencil tutorial system was that it was not possible to provide the students information about why their solution did not match the canonical solution. They were simply sent back to the last time their solution matched the canonical solution. This situation was somewhat ameliorated by the use of very small incremental steps within the tutorial system, but nevertheless, this situation was unfortunate.

Overall, we were satisfied with the performance of the stencil tutorials. Students in general felt comfortable with the tutorials themselves, and the students' abilities to correctly solve the activity problems provides evidence that the students were able to understand the concepts being presented.

During the winter 2012 quarter, we are using the tutorials we have created with a much larger group of students in an introductory programming for non-majors course (with approximately 120 students). Students are (randomly) asked to download either the standard version of Alice (which ships with four standard stencil tutorials that explain how animation works in Alice, but do not attempt to teach programming in Alice) or with a version of Alice that contains our tutorials. Pre- and post-test are being conducted (with respect to Alice programming content knowledge), and students will be asked to fill out surveys at the end of the term investigating whether they used the tutorials, and if so if they found the tutorials helpful.

6. FUTURE WORK

Work is continuing on the construction of the overall ITS. It is intended that the ITS will serve two purposes. The first is for the ITS to help the student to navigate through the tutorials as the student learns to program in Alice. Current work in this area includes designing a way to measure how close students' attempts at solutions to the activities are to a canonical solution. While the problem of determining whether two programs, in general, are equivalent is not solvable, by keeping the activities short, and using various heuristics to determine the similarities between a student's sample solution and a canonical solution. That, coupled with the time it takes for a student to complete an activity will hopefully provide enough information to the ITS to determine whether the student understands the particular topic being taught.

The second area of development for the ITS is in detecting when a student is struggling with a particular programming concept when attempting to solve a larger programming project. A student log system has been created to capture major student actions (and student intended actions) with Alice. It is in the process of being refined so as to be able to detect off-task behavior, but also to provide hint generation. It still needs to be made more robust, and then logs can be analyzed to detect when students are struggling on various concepts. In this vein, the ITS can suggest that a student take a particular tutorial when the system detects that a student is stuck in a particular area of a program.

In the interim, as the tutorials created seem to help students to learn various programming concepts in Alice, we are making the tutorials available to the 3000+ member teacher community. There has been significant interest from the community in getting additional self-guided support materials, both from those engaged in distance education as well as from those who are unable to offer their programming students sufficient in-person laboratory guidance (through open and closed laboratories, and the like).

7. ACKNOWLEDGEMENT

This work is partially supported by NSF award DUE 1021975 - Adding an Intelligent Tutoring System to Alice. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] Anderson, J. R. 1993. Rules of the Mind. Lawrence Erlbaum Associates, Publishers.

[2] Anderson, J.R. and Skwarecki, E. 1986. The automated tutoring of introductory computer programming. Communications of the ACM, 29: 842-849.

[3] Anderson, J. R., Conrad F. G. and Corbett, A. T. 1989. Skill acquisition and the lisp tutor. Cognitive Science 13(4): 467-505.

[4] Baxter, E. 1990. Comparing conventional and resource based education in chemical engineering: student perceptions of a teaching innovation, Higher Education, 19:323-340.

[5] Corbett, A. T. and Anderson, J. R. 1992. Student modeling and mastery learning in a computer-based programming tutor. Proceedings of the International Conference on Intelligent Tutoring Systems.

[6] Cuny, J. 2010. Finding 10,000 teachers. CSTA Voice, 5(6): 1-2.

[7] Gage, M. E. and Pizer, A. K. 1999. WeBWorK – Math homework on the Web. Proceedings of the Annual International Conference on Technology in Collegiate Mathematics.

[8] Gertner A. S. and VanLehn, K. 2000. ANDES: A coached problem solving environment for physics. Proceedings of the International Conference on Intelligent Tutoring Systems.

[9] Gilbert, S., Blessing, S. and Blankenship, L. 2009. The accidental tutor: overlaying an intelligent tutor on an existing user interface. In Proceedings of the 27th international conference extended abstracts on Human factors in computing systems (CHI EA '09). ACM, New York, NY, USA, 4603-4608.

[10] Gotel, O. and Scharff, C. 2007. Adapting an Open-Source Web-based assessment system for the automated assessment of programming problems. Proceedings of IASTED Web-based Education Conference.

[11] Graesser, A. C., VanLehn, K., Rosé, C. P., Jordan, P. W. and Harter, D. 2001. Intelligent tutoring systems with conversational dialogue, AI Magazine, 22(4):39-51.

[12] Grossman, T. and Fitzmaurice, G. 2010. ToolClips: an investigation of contextual video assistance for functionality understanding. In Proceedings of the 28th international conference on Human factors in computing systems (CHI '10). ACM, New York, NY, USA, 1515-1524.

[13] Jin, W. 2008. Pre-programming analysis tutors help students learn basic programming concepts. In Proceedings of the 39th SIGCSE technical symposium on Computer science education (SIGCSE '08). ACM, New York, NY, USA, 276-280.

[14] Kelleher, C. and Pausch, R. 2005. Stencils-based tutorials: design and evaluation. In Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '05). ACM, New York, NY, USA, 541-550.

[15] Knabe, K. 1995. Apple guide: A case study in user-aided design of online help. In Conference companion on Human factors in computing systems (CHI '95), I. Katz, R. Mack, and L. Marks (Eds.). ACM, New York, NY, USA, 286-287.

[16] Koedinger, K. R., Anderson, J. R., Hadley, W. H., and Mark, M. A. 1997. Intelligent tutoring goes to school in the big city, Journal of Artificial Intelligence in Education. 8(1): 30-43.

[17] Lesgold, A., Lajoie, S., Bunzo, M. and Eggan. G. 1992. SHERLOCK: A coached practice environment for an electronics troubleshooting job, in J. H. Larkin and R. W. Chabay (eds.) Computer Assisted Instruction and Intelligent Tutoring Systems, 201-238.

[18] Markoff, J. 2011. Virtual and artificial, but 58,000 want course. New York Times. Available from: http://www.nytimes.com/2011/08/16/science/16stanford.html Accessed 8/31/2011.

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

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

[21] 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.

[22] Sack, W., Soloway, E. and Weingrad, P. 1992. From PROUST to CHIRON: ITS Design as Iterative Engineering: Intermediate Results are Important! In Larkin, J.H. and Chabay, R.W. (Eds.), Computer-Assisted Instruction and Intelligent Tutoring Systems: Shared Goals and Complementary Approaches. Lawrence Erlbaum Associates, Hillsdale, NJ, 239-274.

[23] Shaffer, S. C. 2005. Ludwig: An online programming tutoring and assessment system, ACM SIGCSE Bulletin, 37(2):56-60.

[24] Soh, L. K., Blank, T., Miller, L. D. and Person, S. 2005. ILMDA: An intelligent learning materials delivery agent and simulation. Proceedings of International Electro-Information Technology Conference.

[25] Soh, L. K., and Blank, T. 2008. Integrating case-based reasoning and meta-learning for a self improving intelligent tutoring system. International Journal of Artificial Intelligence in Education, 18: 27-58.

[26] Soloway, E., E. Rubin, B. Woolf, W.L. Johnson and J. Bonar, 1983. MENO II: An AI-based programming tutor. J. Computer-Based Instruction, 10: 20-34.

[27] Khan Academy. http://www.khanacademy.org/ Accessed 8/31/2011.