Alice, BlueJ and StarLogo TNG – Next Generation Environments for Learning Programming

Dixie Margaret Davis
EDF 5906/DIS/Dr. John Keller
August 11, 2008
BACKGROUND

COMPUTER SCIENCE CURRICULUM APPROACHES

TAXONOMY OF PROGRAMMING ENVIRONMENTS

ALICE 2.0

BLUEJ

STARLOGO TNG

CONCLUSION

REFERENCES
**Background**

Learning, working and thriving in a knowledge-based economy demands different skills than those required in the past. Skills that involve technology; skills that allow students to participate in knowledge sharing and social collaboration; and skills that develop higher-order thinking to tackle complex, real world issues are all important if today’s students wish to flourish beyond the classroom (Dede, 2000; Klopfer & Yoon, 2005a). One ramification of tackling complex, real world issues is that the analysis and eventual solution may require working with an abstraction or model of the actual system. Working with models often demands technology and frequently entails integrating data from various sources so that the problem environment and potential solutions can be simulated and visualized. These types of learning and practice activities are necessary to help students successfully perform in the interconnected 21st century world (Dede, 2000).

Since technology is a driving factor in a knowledge-based world, it is understandable that an increased demand for programming skills and a similar uptick in the popularity of introductory programming courses have been seen in the workplace and universities (Robins, Rountree, & Rountree, 2003, p. 137). Programming skills are not easy to acquire, however. Many students drop out of computer sciences programs before moving on to higher level courses (Sykes, 2007, p. 224). Why is programming hard? Researchers posit that it is a number of factors including: the complexity and abstract nature of the task itself; juggling multiple mental models that involve the problem and solution domain (the computer); and, the issue of cognitive load that the programming feat requires (Robins et al., 2003). Programming classes can actually undermine
confidence and students’ attitudes toward programming - “Students leave programming units with very negative attitudes toward programming, with little confidence in their programming abilities, and with little more understanding of programming than what they began with” (Bishop-Clark, Courte, Evans, & Howard, 2007, p. 194).

Due to these challenges, educators and researchers have been interested for the past four decades in developing specialized programming languages and environments that will reduce the complexity of programming and thereby make the subject accessible to a greater number of people (Kelleher & Pausch, 2005, p. 83). These specialized teaching systems mediate some combination of the factors that contribute to the stress novice programmers encounter. For example, some may address the challenge of complexity by simplifying the job of entering code; or, others may include visual components that can help reduce cognitive demand (Cooper, Dann, & Pausch, 2003, p. 20; Kelleher & Pausch, 2005, pp. 85-86). Regardless of the composition, an overall goal of these tools is to help sustain and improve motivation so that more individuals can become proficient at programming (Kelleher & Pausch, 2005, p. 131).

This paper will examine three of these specialized toolsets, specifically the programming environments, and consider the way they individually address the challenges of learning programming. The three software products discussed are: Alice, BlueJ and StarLogo TNG (Carnegie Mellon University; Kolling & Rosenberg; MIT Scheller Teacher Education Program). The designers for each set of these software learning environments saw a deficiency in the current collection of tools and developed a solution to address those needs.
All three software packages discussed here are freely available for download and were developed within university environments for either teaching programming skills or for reducing some of the obstacles to programming ("Alice.org"; "BlueJ - Teaching Java"; “StarLogo on the Web”). All three are still in active production and represent current releases. BlueJ and StarLogo TNG represent very recent releases, and according to the project website, Alice 3.0 is expected to be available in the late 2008-early 2009 timeframe (“Alice.org”). Each package also has corresponding instructional support materials for educators to use in their classrooms and two even have accompanying textbooks (“Alice.org”; “BlueJ - Teaching Java”; “StarLogo on the Web”).

Before beginning the individual examinations, an explanation of computer science teaching strategies and a useful taxonomy of educational programming tools will be presented.

**Computer Science Curriculum Approaches**

The Joint Task Force on Computing Curricula reported in 2001 that the predominant method of teaching introductory programming is a strategy called “programming-first” (Computing Curricula 2001 Computer Science, 2001, p. 22). This strategy maintains that since programming is a fundamental core to the computer science discipline, it should be taught first in the computer science curriculum. The report also discusses three of five approaches (imperative-first, objects-first and functional-first) to implementing the programming-first strategy.
These approaches are represented in Figure 1 below.

![Image of figure 1](image-url)

**Figure 1: Computer Science Course Levels and Implementation Strategies** (*Computing Curricula 2001 Computer Science, 2001, p. 18*).

Two of the learning environments discussed in this paper promote the objects-first strategy: Alice and BlueJ (Cooper et al., 2003, p. 19). The objects-first strategy concentrates on programming, but as it relates to object-oriented principles and design (*Computing Curricula 2001 Computer Science, 2001, p. 30*). The report maintains that an advantage to this approach is that it provides students with an early chance to interact with object-oriented programming, “which has become increasingly important in both academia and industry” (*Computing Curricula 2001 Computer Science, 2001, p. 30*). Object-oriented programming has gained in popularity since the 1990s and promotes a programming model that focuses on objects, messages and hierarchies of objects to create software applications (“Object-oriented programming - Wikipedia, the free encyclopedia”).

The report authors maintain, however, that an object-oriented approach is not necessarily an approach without disadvantages because object-oriented languages, like Java and C++, add complexity, rather than reduce it (*Computing Curricula 2001 Computer Science, 2001, p. 30*).
One set of researchers studying the effectiveness of the Alice software offer this reason for the complexity of object-oriented (OO) languages:

    OO languages require that students not only learn the material for an imperative language core (e.g., assignment, decisions, functions, procedures, repetition, arrays) but also learn the additional concepts of class, object, information hiding, inheritance and polymorphism” (Moskel, Lurie, & Cooper, 2004, p. 75).

An objects-first approach may be considered a more comprehensive approach, however, as it requires an incorporation of much of the concepts, specifically state and functions, which are the focus in the other two approaches (Cooper et al., 2003, p. 19).

**Taxonomy of Programming Environments**

Two members of the Alice development team created a useful taxonomy for organizing programming environments and languages that are focused on novice programmers (Kelleher & Pausch, 2005, p. 83). Two of the three environments discussed in this paper are present in the taxonomy, but StarLogo TNG was released after the publication of the taxonomy and is not present in the structure. Its predecessor, StarLogo, is represented, however, and its placement is used as a surrogate because the goals have not changed (“StarLogo on the Web”). The taxonomy presents 86 different environments and languages, which are divided initially by their primary goal: either to teach programming, or to programming to “empower their users” (Kelleher & Pausch, 2005, p. 84). Alice and BlueJ fall into the first group: “Teaching Systems” and StarLogo TNG is a member of the second group, “Empowering Systems”, as its users employ programming primarily as a means to model complex systems, such as epidemics, or the behavior of ant colonies (Kelleher & Pausch, 2005, pp. 85-86).
The next division focuses on the particular aspect of learning to program that the system attempts to clarify or simplify. Concentrating on Alice and BlueJ first, these two teaching systems both are part of the group that focuses on the “Mechanics of Programming”. Members of this group make it easier for novice programmers to either provide instructions to the computer (“Expressing Programs”); or, they make it easier for beginners to structure their code (“Structuring Programs”); or, they make it easier for beginners to understand how the computer executes the code (Kelleher & Pausch, 2005, p. 87). Alice and Blue J fall into the first two divisions, respectively.

<table>
<thead>
<tr>
<th>Teaching systems</th>
<th>Alice 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanics of Programming</td>
<td></td>
</tr>
<tr>
<td>Expressing Programs</td>
<td></td>
</tr>
<tr>
<td>Find Alternatives to Typing Programs</td>
<td></td>
</tr>
<tr>
<td>Construct Programs Using Graphical or Physical Objects</td>
<td></td>
</tr>
<tr>
<td>BlueJ</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Teaching systems</th>
<th>BlueJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanics of Programming</td>
<td></td>
</tr>
<tr>
<td>Structuring Programs</td>
<td></td>
</tr>
<tr>
<td>Making New Models Accessible</td>
<td></td>
</tr>
</tbody>
</table>

The programs that make it easier for novice programmers to provide instructions or “express” their intentions are further divided by how they make this process easier. They either simplify the language that programmers use to communicate with the computer, or as is the case with Alice, they provide an alternative way to communicate with the computer (Kelleher & Pausch, 2005, p. 87). The final grouping for Alice is how that program allows users to construct instructions- it employs a strategy of using graphic elements that represent the main components of a program, such as “commands, controls structures, or variables” (Kelleher & Pausch, 2005, p. 91).

As mentioned earlier, BlueJ is represented in the taxonomy as a programming environment that makes it easier for learners to structure code. The sub-group within which BlueJ belongs focuses
on making new modes of programming, specifically object-oriented programming, simpler and more understandable to beginners (Kelleher & Pausch, 2005, pp. 100-101).

The StarLogo TNG environment as indicated previously is primarily a modeling and visualization tool that employs programming. These “Empowering systems” focus on helping their users build things (Kelleher & Pausch, 2005, p. 109). StarLogo TNG is a part of a sub-division of these systems that create opportunities for learners to explore specific domains (Kelleher & Pausch, 2005, p. 122). In the case of StarLogo TNG, the system focuses on the domain of education and it allows users to build and explore complex models, systems and processes, such as evolution, erosion and enzyme reactions (Kelleher & Pausch, 2005, p. 125; MIT Scheller Teacher Education Program).

![Empowering systems
Activities Enhanced by Programming
Education
StarLogo TNG](unnamed)

Figure 3: The Taxonomic Divisions of StarLogo TNG.

The three programming systems discussed in this paper have evolved from previous releases (“Alice.org”; “BlueJ - Teaching Java”; “StarLogo on the Web”). StarLogo TNG and Alice are derivatives of one of the most influential educational programming system – Logo 1967 (Kelleher & Pausch, 2005, p. 126). Logo is a functional system developed at MIT (“The Logo Programming Language,” 1997). BlueJ is a derivative of Smalltalk which was developed originally in the 70s as an object-oriented educational programming language (“BlueJ - Wikipedia, the free encyclopedia,” 2008). Alice is also influenced by Smalltalk and the
Alternate Reality Kit developed in 1987, which was an early system that allowed users to interactively build simulations in a visual environment (Kelleher & Pausch, 2005, p. 111).

Alice 2

![Figure 4: Screen capture of Alice 2.](image)

Alice was created by a research group at Carnegie Mellon University, under the direction of the late Randy Pausch (“Alice.org”). The system allows novice programmers an opportunity to create animated 3D virtual worlds that can be played as a movie or a game (Kelleher & Pausch, 2005, p. 94).
As stated by the program team, Alice has three goals: “Reduce the complexity of details that the novice programmer must overcome, provide a design first approach to objects, visualize objects in a meaningful context” (Cooper et al., 2003, p. 20).

As mentioned in the taxonomy portion of this paper, Alice is a teaching system that enforces the mechanics of programming by providing users with a different option for constructing code (Kelleher & Pausch, 2005, p. 91). Students develop programs by selecting graphic elements that represent functional commands and methods that an object can perform. These elements are then dragged into a code editor where properties and parameters can be set from dropdown menus. Actions and properties are specified in simple language and lines of code can be played in the form of a short movie or interactive game. Members of the development team argue that this type of code generation helps simplify the process of programming: “The mechanism for generating code relies on visual formatting rather than details of punctuation. The gain from this no-type editing mechanism is a reduction in complexity” (Cooper et al., 2003, p. 21).

Alice distinguishes itself from the rest of the code generation programming systems by addressing all the key constructs (e.g., syntax, variables, I/O, conditionals) that are recommended inclusions in introductory programming courses (Kelleher & Pausch, 2005, p. 95; Computing Curricula 2001 Computer Science, 2001, p. 89).

Alice also offers dramatic help to the challenges of visualization via its 3D animated environment. For example, since students can watch skaters jump, penguins sing and palm trees
sway, concepts associated with object-oriented programming, are rendered in a more meaningful and perhaps more relevant manner (Cooper et al., 2003, p. 20).

Finally, Alice addresses the need to enhance and sustain students’ motivation for programming by allowing them to be productive almost immediately by using virtual worlds pre-packaged with the environment. Rather than creating the traditional “Hello, world!” first program so common with other beginning programming tools, Alice enables the student to start developing stimulating and compelling interactions that can involve amusement parks, flight simulators or even lunar modules (Bishop-Clark et al., 2007, p. 194). Quantitative research studies have cited improvements in student achievement and one study has reported increases in enjoyment, confidence and understanding relating to programming concepts (Sykes, 2007; Cooper et al., 2003; Moskel et al., 2004; Bishop-Clark et al., 2007).

A discussion on any software program would not be complete without addressing any shortcomings or weaknesses. One developer from the project team admits that one of its greatest strengths could also be classified as a weakness—“Students do not develop a detailed sense of syntax” (Cooper et al., 2003, p. 22). Students involved in one of the research efforts cited another weakness—Alice is not equipped to address “real-world” problems like Java and C++ (Sykes, 2007, p. 240). For example, students cannot create their own 3D objects within Alice, they are limited to the 3D objects provided within the program.
BlueJ uses the Java programming language and a focused interactive development environment to teach introductory object-oriented principles (“BlueJ - Teaching Java”). It was developed by Michael Kolling and John Rosenberg specifically to support the needs of first year programmers (Kolling, Quig, Patterson, & Rosenberg, 2003, p. 264).

Michael Kolling and John Rosenberg evaluated several programming environments focused on novice programmers and found that the majority were not object-oriented, were often too
complex and focused too heavily on the user interface (Kolling et al., 2003, p. 250). Creating a tool that addressed these needs then became BlueJ’s goals.

As mentioned previously in the section on taxonomy, BlueJ is classified as a learning environment that helps students with the structure of code while exposing and simplifying the principles of object-orientation (Kelleher & Pausch, 2005, pp. 100-101). BlueJ users are typically presented with existing classes which are rendered graphically and whose relationships are depicted hierarchically (Kolling et al., 2003, p. 252). Class structure is depicted visually which can help with some of the difficult, abstract concepts of object-orientation (Kolling et al., 2003, p. 256). The use of graphics to visually display the class structure is also a design element that addresses the challenges novice programmers generally have with object visualization.

Complexity is reduced within BlueJ by allowing some code creation via mouse-clicks. Learners create objects by clicking on the class graphic and selecting its constructor method. Right-clicking on objects display their available methods and allow the learners an opportunity to inspect properties associated with the class, such as its name or year of birth in the case of a Student object, for example (Kolling et al., 2003, p. 254). Source code is available by double-clicking on a class and code can be modified, debugged and re-compiled within the environment. Rather than start from scratch, the BlueJ authors point out that due to the inherent “complexity of object-oriented design, it is favorable for novices to start with partially/completely developed projects” (Cooper et al., 2003, p. 22).
BlueJ also addresses the need for simplicity by removing certain aspects that can typically confuse the beginning Java programmer. For example, classes can be instantiated and tested without the normal requirement of a “main” method. Therefore, to be productive, beginning Java programmers do not need to create an entire program first (Kelleher & Pausch, 2005, p. 101).

The developers of the BlueJ system approach the need to develop and sustain motivation in novice programmers by advocating problem-driven and apprentice-based approaches as the instructional strategies that accompany the system. They maintain that programming tasks associated with these approaches “are more likely to generate a sense of excitement and motivate further investigation” (Kolling & Barnes, 2004, p. 286). The authors explain that introductory programming classes should include tasks that build on large projects that have real-world relevance, such as building a calculator or developing an adventure game (Kolling & Barnes, 2004; Kolling et al., 2003).

One weakness that has been reported about the BlueJ system can be partially attributed to the general challenges of teaching object-oriented principles. The authors note that they have received feedback that some students were weaker in areas concerning standard programming constructs, such as data structures and algorithms, though they achieved object-oriented performance objectives (Kolling et al., 2003, p. 265). They maintain this weakness is external to the system and should be addressed by instructors since objects-first approaches require an inclusion of standard programming constructs (Computing Curricula 2001 Computer Science, 2001, p. 89; Kolling et al., 2003, p. 265).
StarLogo TNG

Figure 6: Screen capture of StarLogo TNG.

StarLogo TNG is a learning environment created by the MIT Teacher Education Program that, like Alice, allows students to create 3D worlds (“StarLogo on the Web”). The distinction between the two, however, is the purpose of the programming tool. The worlds developed with StarLogo TNG are primarily used to visualize and educate students on the behavior of complex models and systems, rather than the mechanics of programming (Kelleher & Pausch, 2005, p. 124). As mentioned earlier, this is the main justification for taxonomically classifying StarLogo TNG as an “empowering”, rather than “learning”, system (Kelleher & Pausch, 2005, p. 84).

The goals of StarLogo TNG, as expressed by the authors of the system, are similar to the other two environments already explored. The authors enumerate three goals for the environment: “to
lower the barrier to entry for programming”; to attract more young programmers by providing tools that can create games; and, by creating “compelling 3D worlds that encompass rich games and simulations” (MIT Scheller Teacher Education Program).

The StarLogo TNG environment deals with complexity in a different manner than Alice and BlueJ because technically it is not an environment that directly teaches programming skills. Rather, this environment mitigates the issues of complexity by giving students the opportunities to develop models of complex systems and create simulations that would not be possible without the tools of computer technology. Students work directly with abstractions of complex systems by manipulating the parameters of model algorithms and then analyzing the output of their work (Klopfer & Yoon, 2005b, p. 37). StarLogo TNG users participate in “agent-based modeling” which allows them to develop the rules “that control thousands of objects and observe patterns that arise as a result of these rules.” (Kelleher & Pausch, 2005, p. 125; “StarLogo on the Web”).

Though not directly a programming learning environment, the code required to run StarLogo TNG models, simulations or games is generated from graphic elements like the Alice environment. The structure of the program is visualized in a schematic process map that breaks down the functional units of the program in the upper right hand corner of the editor window, which is called the “StarLogoBlocks” window. Blocks that represent command functions are dragged onto this editor and objects interact with these commands. Programs are then run and displayed as animated movies, simulations or interactive games within the “SpaceLand” window.
Motivation and the compelling visual nature of the StarLogo TNG software are strongly linked. The “SpaceLand” window provides the option to view multiple perspectives and provides a motivating game and play quality similar to Alice. Students can be productive right away and start gaming and generating simulations due to several projects that come installed with the software like the pre-built worlds within Alice (“StarLogo on the Web”). One of the examples exhibited on the project site and available from the system shows the effects on members of a marine ecosystem when either one of the members (fish or plankton) is over or under-populated (“StarLogo on the Web”). Unlike Alice, students can generate their own 3D objects, by importing 3D models that are in a specified, compatible format (“StarLogo on the Web”).

StarLogo TNG is a very recent release (version 1.0) and, as such, documented weaknesses are difficult to find. Combing the software’s associated websites, it was discovered that there is no formal structure yet for changing the locale (or language) of the software (“StarLogo on the Web”).

**Conclusion**

Increasingly students must either use technology or develop it to address and understand the complex problems associated with today’s world. To prosper in a work place that requires skills to abstract, model and visualize interconnected systems, students must learn how to program or learn how to use tools that develop programs (Dede, 2000; Robins et al., 2003).

Developing these skills is not elementary, however, and several programming tools exist to help flatten the steep curves students encounter when trying to learn how to produce software
(Kelleher & Pausch, 2005). The latest computer science curricula recommendations actually separate software engineering out as a separate discipline (Computing Curricula 2005 - The Overview Report, 2005, p. 33). The report authors predict a need for universities and other educational institutions to create more software engineering programs (units) and they envision these software engineering programs (units) will provide an “opportunity for an institution to distinguish itself in ways that directly addresses academic and professional challenges” (Computing Curricula 2005 - The Overview Report, 2005, p. 38). This prediction bodes well for future students who will require programming tools that provide even greater innovation in the areas of reducing complexity, improving visualization capabilities and sustaining high level of motivation.
References


"Don't be too timid and squeamish about your actions. All life is an experiment. The more experiments you make the better."

– Ralph Waldo Emerson