Bridging Educational Programming and Production Languages

Jeremiah Blanchard
Dept of CISE
University of Florida
Gainesville, FL 32611, USA
jblanch@cise.ufl.edu

Christina Gardner-McCune
Dept of CISE
University of Florida
Gainesville, FL 32611, USA
gmccune@ufl.edu

Lisa Anthony
Dept of CISE
University of Florida
Gainesville, FL 32611, USA
lanthony@cise.ufl.edu

Abstract
Computer science skills are ever more important in modern society as computers become integrated into practically all occupations and professions. Researchers have been developing learning tools, particularly educational programming environments, to facilitate the learning of computer science concepts at younger ages. These tools have made significant gains in engaging a younger audience and making programming more accessible by incorporating visual elements, drag-and-drop program construction, and media-rich environments. Some platforms are friendly for young children, but lack programmatic complexity, while others lack the accessibility necessary for learners in grades K-8. Thus, current tools *lack a unified bridge* from early educational to practical, applied programming environments. An age-appropriate bridge could facilitate transfer of knowledge and skills by applying techniques shown to encourage such transfer in learning science research.

Author Keywords
Computer Science Education, programming environments, programming languages, transfer.

ACM Classification Keywords
K.3.2 Computers and Information Science Education.

Introduction

Algorithmic thinking and reasoning are becoming more important over time, even for non-computer scientists, as we become more dependent as a society on computer technology [1]. There is also an enormous need for software developers that is expected to continue to grow [2], and basic programming skills are desirable in many technical and non-technical fields (science, business analytics, animation, and design, for example). Educational programming environments – especially those focused on facilitating the learning of novice programmers – provide an important path into the discipline of computer science for learners. The designers of these environments have aimed to make their platforms accessible to lay audiences and particularly children [3]. By making programming more accessible, such environments can facilitate interest in computer science and programming in school-age children [4], and skills learned can transfer to languages used in production settings [5]. Currently, most such environments focus on elementary and middle school students (e.g., Scratch) or late high school and college freshmen students (e.g., Alice), and transfer does not always occur when switching contexts [6]. If constructed mindfully, a bridge that facilitates transfer from childhood educational environments to practical, applied languages could be established. This would facilitate growth of a larger and more qualified candidate pool to address the growing need for individuals skilled in programming.

Background

Work to establish environments to encourage computational thinking and programming dates back most notably to the design and development of Logo by Seymour Papert and his colleagues [7]. In developing Logo, Papert argued that procedural thinking can benefit children by encouraging *metacognition* (thinking about thinking), in addition to enabling programming in a computer-rich world, and sought to create a math environment where students could explore and learn mathematics through programming in a way that is easy to pick up [3]. Current work continues building on Papert's vision; we detail 3 of the most popular tools.

Scratch

Scratch began in 2003 and is developed at the Lifelong Kindergarten group of the MIT Media Lab. Its creators include former members of Papert's MIT Logo Lab. It was designed as a networked programming environment built to reach economically disadvantaged youth at after-school centers with a focus on the 8 to 16 age range [8]. Scratch's interface uses graphical drag-and-drop statements, variables, and control structures (Figure 1a). Scratch also incorporates color-coded and distinctly shaped programming blocks. Scratch has been shown to engage at-risk city youth, encouraging expression and creative thinking [9].

Alice

The Alice project began in 1995 as a research initiative focused on 3D environments at the University of Virginia [10]. Alice focuses on late high school and early college age groups [8]. By the year 2000, Alice was being used to teach introductory programming concepts [11]. A branch known as *Storytelling Alice* (and later *Looking Glass*) was also developed to engage girls in programming [12]. Like Scratch, Alice uses a graphical, drag-and-drop object-oriented language interface with a scene display (Figure 1b), and program execution results are viewable in the scene without compilation. Alice 3 introduced several changes to

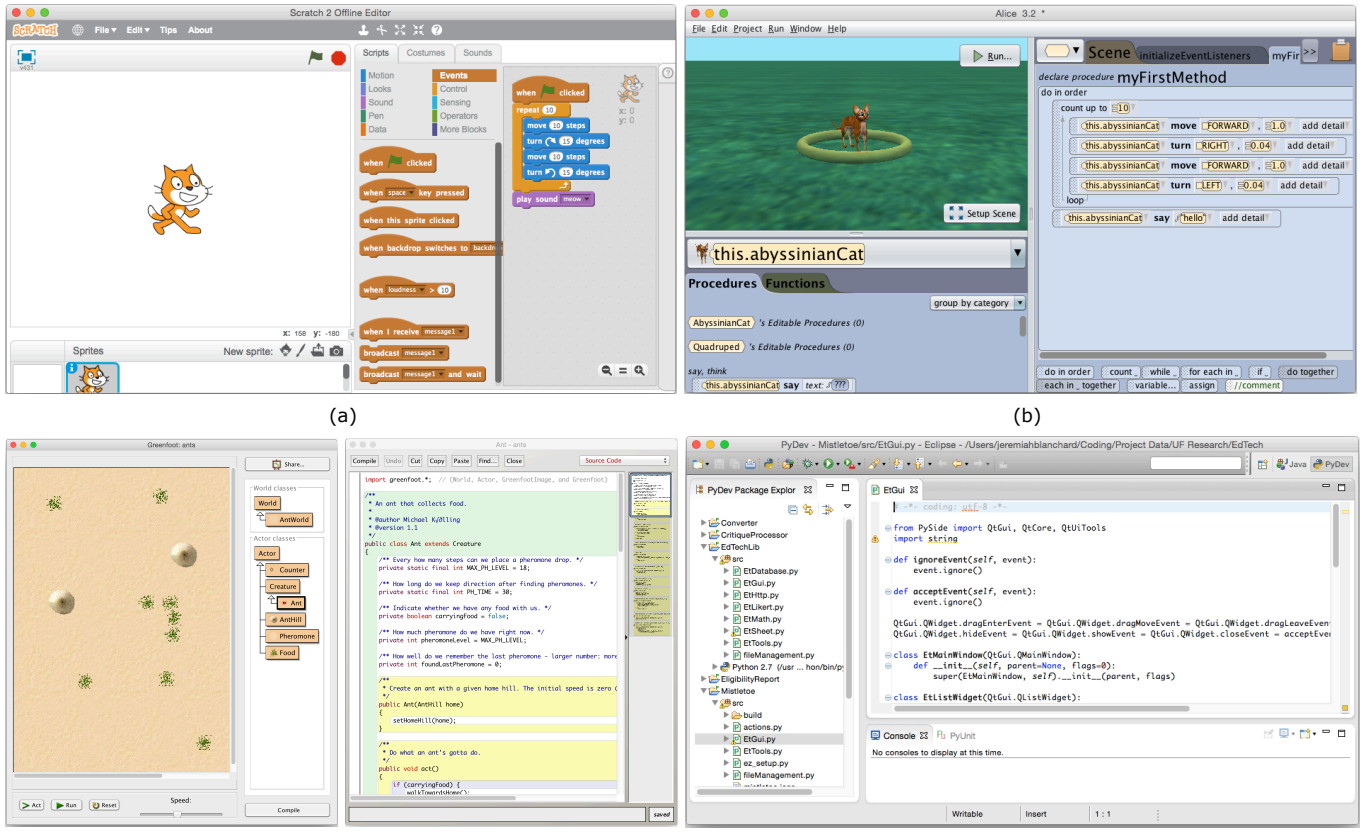

Figure 1: Screenshots of (a) Scratch, (b) Alice, (c) Greenfoot, and (d) Eclipse IDEs.

facilitate Java language learning that have yielded improvements in performance in early computer science (CS1) courses [5]. The Alice project is ongoing.

Greenfoot

Greenfoot has been developed at the University of Kent since 2005 [13]. Greenfoot differs from Alice and Scratch in that it uses both graphical and textual modes for source code entry. Class relationships and scene object positions are manipulated using a mouse-driven interface, while class definitions are written directly in Java code (Figure 1c). Greenfoot also incorporates a debugger and is targeted for ages 14 and older.

Bridging

Although these tools have shown great promise in exposing younger audiences to computer science and computational thinking concepts, what is still lacking is that *bridge* from the simplified and abstracted languages and tools targeted to K-8 students to more advanced, complex environments. These more advanced environments have shown success at the high school and college levels in transitioning students to production programming languages used by programmers today such as Java and the more complex IDE tools used for these languages (e.g., Eclipse, Figure 1d). In addition, many transition tools (like Greenfoot) target Java, whose syntax may be more difficult for beginners than some alternatives [14]. A bridge would facilitate transfer of knowledge and skill from the early educational environments to an applied one while remaining accessible. This bridge could be explicit scaffolding that facilitates movement from existing educational environments to production languages, or a completely new environment developed explicitly to grow with students as their cognitive abilities mature. We propose that the robust and effective development of such a bridge presents a key research challenge of making "every child a coder."

References

1. Wing, J.M. Computational thinking. *Communications of the ACM 49*, 3 (2006), 33-35.
2. U.S. Department of Labor, Bureau of Labor Statistics. 2014. Occupational outlook handbook, 2014-15: Software developers.
3. Papert, S. Mindstorms: children, computers, and powerful ideas. Basic Books, Inc., 1980.
4. Nikiforos, S., Kontomaris, C., and Chorianopoulos, K. MIT Scratch: A Powerful tool for improving teaching of programming. In Proc. *5th Conference on Informatics in Education 2013,* (2013).
5. Dann, W. et al. Mediated transfer: Alice 3 to Java. In *Proc. SIGCSE 2012*, (2012), 141–146.
6. Perkins, D.N. and Salomon, G. Transfer of Learning. *International encyclopedia of education 2nd ed*, Pergamon Press (1994), 6452–6457.
7. Logo Foundation. What is Logo? Retrieved March 29, 2015 from http://el.media.mit.edu/logo-foundation/logo/index.html.
8. Utting, I., Cooper, S., and Kölling, M. Alice, greenfoot, and scratch--a discussion. *ACM TOCE 10*, 4 (2010), 1–11.
9. Peppler, K.A. and Kafai, Y.B. Collaboration, computation, and creativity: Media arts practices in urban youth culture. In *Proc. CSCL 2007*, (2007), 586–588.
10. Pausch, R. et al. A brief architectural overview of Alice, a rapid prototyping system for virtual reality. *IEEE Computer Graphics and Applications Magazine*, 1995.
11. Cooper, S., Dann, W., and Pausch, R. Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges 15*, (2000), 107–116.
12. Kelleher, C., Pausch, R., and Kiesler, S. Storytelling alice motivates middle school girls to learn computer programming. In *Proc. CHI 2007*, (2007), 1455-1464.
13. Kölling, M. and Henriksen, P. Game Programming in Introductory Courses with Direct State Manipulation. *ACM SIGCSE Bulletin 37,* 3 (2005), 59-63.
14. Agarwal, K., Agarwal, A., and Celebi, M. Python puts a squeeze on java for CS0 and beyond. *Journal of Computing Sciences in Colleges 23*, (2008), 49–57.