A Comparison of Different Approaches to the Introductory Programming Course

Michael Goldweber[1] **Joe Bergin**[2] **Raymond Lister**[3] **Myles McNally**[4]

[1] Xavier University
Cincinnati, Ohio, USA 45207
Email: mikeyg@cs.xu.edu

[2] Pace University
New York, New York, USA 10038
Email: berginf@pace.edu

[3] University of Technology
Sydney, Australia NSW 2007
Email: raymond@it.uts.edu.au

[4] Alma College
Alma, Michigan, USA 48801
Email: mcnally@alma.edu

Keywords: Introductory programming environments, microworlds, robotics.

1 Introduction

Over the last few years the number of visual and non-traditional programming environments to support the introductory programming course has greatly increased. While in the past, instructors of the introductory programming course simply had to select a compiler (and possibly a development environment), today they face a plethora of choices. This panel will present four such popular visual/non-traditional programming environments.

A common thread among these environments is that they all seek to continuously illustrate/visualize the complete program state throughout program execution. The underlying pedagogic assumption is that such continuous non-textual visual feedback is superior for introductory students than more traditional programming languages/environments; where program state feedback is strictly textual, limited in scope, and periodic (e.g. cout statements). This panel will not only present four such popular programming environments, but, since each approach has its own strengths and weaknesses, engage the audience in a discussion comparing and contrasting them as well.

2 Lego Mindstorms - Myles McNally

Lego Mindstorms is an inexpensive robotics system consisting of a microprocessor brick, various sensors and motors, and numerous Lego pieces. It can be programmed in a variety of languages, including Java and C++. Since its initial release in 1996 dozens of papers have been written on its use in computer science education (Barnes 2002, Kay 2003, Lawhead, Bland, Barnes, Duncan, Goldweber, Hollingsworth & Schep 2003). Most describe its use, in one way or another, in laboratory settings. Numerous workshops on using the Mindstorms platform have been held, attended by hundreds of faculty. Yet the platform's use is not as widespread as this level of interest would suggest. I argue this is because of a number of easily countered misconceptions, such as:

- **It's a toy, you can't teach real programming with it.** Yes it's a toy, but an educational toy. And you can teach real programming with it. Laboratory experiences have been developed for Mindstorms on everything from the difference between local and global variables to interfaces, inheritance, and polymorphism.
- **Adequate laboratory materials do not exist.** They say you can find everything on the web, and a large number of papers and laboratory exercises for Mindstorms can be so found. I am part of a project that is building a comprehensive set of laboratory materials for using Mindstorms throughout the CS undergraduate curriculum (Klassner, Lawhead & McNally 2005). Among the items we have developed is a set of twelve laboratories for an object oriented introductory course, each augmented by video clips of task solutions.
- **The kits are too expensive.** You can buy ten kits for the price of a computer workstation, and using pair programming that would be sufficient for a typical lab. That's not so much.
- **Why not just use a simulator?** In fact other members of this panel are going to suggest something like that. The reasons are numerous: Students enjoy working with the robots, leading to more time on task. Robots literally embody state and behaviour, physically modelling the structure of the programming solutions. Students receive immediate visual feedback and enjoy interacting with robots, naturally leading to increased interaction with other members of the class as well.

Conclusion: Mindstorms-based robotics are a great choice for CS 1.

3 Karel J Robot - Joe Bergin

Karel J Robot (Bergin, Stehlik, Roberts & Pattis 2005) is a microworld that uses a small subset of Java to teach fundamental computing concepts, especially object-oriented programming. The language of Karel is simply the language of message passing with everything else left out. A rich metaphor is presented to students that does not depend on students understanding more primitive computational ideas, such as the Von Neumann architecture. This approach has many advantages. First, is that the syntactic and conceptual load on students is much reduced, yet the power of the language is still that of a Turing Machine. Second is that the system is very visual, with feedback to students given immediately as a program executes. Being Turing complete, the language supports an algorithmic rich first course, while still teaching object-oriented (polymorphic) programming.

The beauty of message passing as a core language is that every time you write a new class you extend the language, but don't extend the syntactic load on the students. This means that student effort is spent in solving a rich set of exercises, and not in learning language details. It does require that some topics typically taught early must be delayed. Karel does not teach Java's arithmetic operations or primitive data types, though arithmetic algorithms are included in the examples and exercises. Experience has shown Karel J Robot to be an effective introduction to computing, taking only a few weeks to master. Students then have a solid picture of computing, good problem solving skills, and a rich base from which to learn other, more traditional, topics.

4 Alice - Michael Goldweber

The Alice programming environment (Stage3 Research Group 2005) is an interactive 3D microworld designed to facilitate the learning of computer programming by large portions of the general population. Alice is programmed using an iconic approach; Alice programmers do not type commands which must adhere to a strict syntactic model, but instead use a drag-and-drop interface to manipulate objects. While Alice is based on an object world-view, it stresses storytelling as its primary metaphor.

Alice was designed to overcome the perceived sociological and mechanical barriers which prevented middle school girls from being successful computer programmers. In addition to its growing use at the middle/high school level, Alice has proven useful for introducing programming at the university level (Cooper, Dann & Pausch 2003) in

- non-majors (CS 0) courses,
- breadth-first introductory courses,
- traditional programming-first introductory courses.

The Alice environment, unlike most other microworlds, provides students with a very rich set of objects for manipulation. Students can not only create their own methods and functions for these objects, but can also create their own objects. Alice also explicitly supports the event-driven programming model as well. Unfortunately, Alice objects do not support inheritance, polymorphism, or data structures more complex than single-dimensional arrays.

Alice's primary strength is the ease with which introductory students can be almost immediately successful. Since there is no syntax to master and the iconic interface allows the programmer, at any step in the program, to only select legal choices/statements/commands, every program works - it just might not do what was intended/solve the problem at hand. This allows for deep and sophisticated self-directed student exploration; something that syntax-based languages and their concomitant compiler wrestling matches implicitly discourages.

5 PigWorld - Raymond Lister

PigWorld (Lister 2004) is a microworld for introducing students to Java programing. Boy and girl pigs move around a maze, finding food, avoiding predators, and - when the opportunity presents itself - making love.

Pigworld was designed as an attempt to transcend the dialectic of "procedural first" vs. "objects first". Many of the proponents of "procedural first" argue that because objects contain methods which in turn contain procedural code, it follows that algorithms written in the procedural paradigm should be taught first. The argument is based on a reductionist "inside out" view of algorithms. While it is true that, in the procedural style, algorithms are encoded explicitly within the methods of an object, in object-oriented code complex algorithms can emerge from a sequence of simple interactions between objects. For example, in PigWorld, boy and girl pigs find each other in the maze by a series of simple object interactions which collectively implement Dijkstra's shortest path algorithm. With PigWorld I have taught Dijkstra's shortest path algorithm to students in their first semester of programing, whereas no teacher of the procedural paradigm would dream of teaching such an algorithm so early.

PigWorld also demonstrates that linked-lists are the natural first data structure for students being taught objects-first. What more natural way could there be of reinforcing the

1. principle of message passing, and
2. difference between a class and an object,

than by having a series of objects of the same class act as the nodes of a linked list? In objects-early, arrays should be taught after linked lists. In fact, arrays should not be taught at all in the first semester. Arrays are a low-level machine-oriented concept. The teaching of arrays to novice programmers is destined for the same oblivion as the teaching of assembler.

Many of the reported problems of teaching objects-early are in fact due to misguided attempts to teach "Objects Etcetera" - the teaching of objects-oriented programming while maintaining the traditional procedural content of a first course in programming. PigWorld demonstrates that an interesting pedagogic microworld requires a minimum of procedural programming.

References

Barnes, D. (2002), 'Teaching introductory java through lego mindstorm models', *ACM SIGCSE Bulletin* **34**(1).

Bergin, J., Stehlik, M., Roberts, M. & Pattis, J. (2005), Karel J. Robot: A gentle introduction to the art of object oriented programming. Accessed 18 Aug 2005.
URL: *"http://csis.pace.edu/ bergin/KarelJava2ed/"*

Cooper, S., Dann, W. & Pausch, R. (2003), Teaching objects-first in introductory computer science, *in*

'Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education'.

Kay, J. (2003), 'Teaching robotics from a computer science perspective', *Journal of Computing Sciences in Colleges* **19**(2).

Klassner, F., Lawhead, P. & McNally, M. (2005), LMICSE: Lego mindstorms in computer science education project. Accessed 18 Aug 2005.
URL: *"http://www.mcs.alma.edu/LMICSE"*

Lawhead, P., Bland, C., Barnes, D., Duncan, M., Goldweber, M., Hollingsworth, R. & Schep, M. (2003), 'A road map for teaching introductory programming using lego mindstorms robots', *ACM SIGCSE Bulletin* **35**(2).

Lister, R. (2004), Teaching java first: Experiments with a pigs-early pedagogy, *in* 'Sixth Australasian Computing Education Conference (ACE2004)'.

Stage3 Research Group (2005), Alice: Free, easy, interactive 3d graphics for the www. Accessed 18 Aug 2005.
URL: *"http://www.alice.org/"*