

POOPLES - PRE-OBJECT ORIENTED PROGRAMMING LEARNING ENVIRONMENTS

Fintan Culwin
CISE
London South Bank
University
London SE1 0AA
fintan@lsbu.ac.uk

Kemi Adeboye
CISE
London South Bank
University
London SE1 0AA
adeboyk@lsbu.ac.uk

Phil Campbell
CISE
London South Bank
University
London SE1 0AA
campbep@lsbu.ac.uk

<http://cise.lsbu.ac.uk/pooples>

ABSTRACT

A need for pre-object oriented programming instructional environments is proposed and several existing possibilities are evaluated and found wanting. The design, implementation and operation of three Pre-Object Oriented Programming Learning Environments (POOPLES) is presented; together with the outcomes of initial evaluations.

Keywords

pre-programming, object first, microworlds, learning programming

1. INTRODUCTION

This section describes the motivation behind the production of a series of Pre-Object Oriented Programming Learning Environments, known as POOPLES. Previous work by the authors with learning objects such as the Java Class Factories (JCF) [1] had reminded them of the complexity inherent in the initial stages of learning object oriented software development. Although tools such as the JCF provided a learning bridge over the complexities involved in the production of Java source code; they did little or nothing to introduce and ground the essential concept of software object.

The fundamental syntactic concept of object can be expressed in the canonical:

object.method(arguments);

Which is also known, since Smalltalk, as the colloquialism of sending an object a message.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

© 2006 Higher Education Academy

Subject Centre for Information and Computer Sciences

Associated with this fundamental concept are ancillary concepts such as instantiation (aka construction), state, behaviour and identity; as well as procedural underpinnings such as selection and iteration.

Pre-programming is the process of identifying the intellectual and practical pre-requisites for starting to learn programming and devising activities to ensure that they are present in students. It has become less fashionable than it once was. Consequently the resources available for pre-programming [2,3] are not directed at, and so not appropriate for, the object oriented paradigm as they do not prepare for the appropriate idioms and colloquialisms.

Four significant possible resources were identified by the team, a Java implementation of Karel the robot, Jeroo, Alice and ToonTalk. Karel the robot [4,5] is well known but was deemed unsuitable as it lacks the granularity of an effective learning object in that it attempts to provide a richly featured programming environment including for example tasking. Jeroo [6,7] has an appropriate degree of granularity, restricting itself to a single simple simulated microworld. However its syntax, whilst respecting the OO idiom, did not have the true feel of Java and this was felt to present a possible barrier to effective transfer of the learning outcomes. Alice [8, 9] is also a large scale microworld with non-Java like syntax. ToonTalk [10, 11] is aimed at much younger students and is also large scale and non-Java.

At the same time as the authors were considering OO pre-programming learning objects, they evaluated and later adopted for first year use Dr Java [12] a simple Java IDE. One of the features of Dr Java that was felt to be particularly useful was the interactions pane. This provides an interpreter capability allowing Java expressions and clauses to be typed in and immediately executed; avoiding the compile run cycle. The major advantage of interpreters for neophyte developers is that there is less of a gulf of execution between intention, expression and outcome. Both Karel and Jeroo

were compiler based environments. Alice allows direct manipulation via object inspection and also scripting in the python language. ToonTalk has an extensive interactive metaphor for programming largely by drag and drop demonstration.

Several further pedagogic consideration informed the decision to build POOPLES rather than reuse or adapt existing environments. One was to triangulate the student's experience by building a number of related simple simulated microworlds. The intention being that one of them could be used for purposes of exposition and demonstration, whilst the others could be used for exploratory learning or, possibly, for summative assessment. The primary pedagogic intention however was to aid the process of accommodation by presenting alternative viewpoints.

Karel the robot and Jeroo are relatively abstract entities moving in abstract Cartesian space. Given the learned helplessness regarding mathematical abilities that seems inherent in most of our students, we wished to avoid Cartesian concepts and,

furthermore, to ground the entities in more concrete environments. Both Alice and ToonTalk have realistic and extensive microworlds. However they lack any particular task structure, although one can easily be programmed into instances of them.

Observations of students in the first stages of learning software development had led the team to the conclusion that the keyboard was a major impediment to effective learning. The complexities of correct composition of program statements and the mysterious messages produced when the inevitable mistakes were made lead to frustration and disengagement. Accordingly 'programming by pushing buttons' [13] was designed for POOPLE operation. The design and implementation of this facility will be described below.

Finally a game element was introduced into the environment in the expectation that this might provide further cultural context to the activity and add a fun factor.

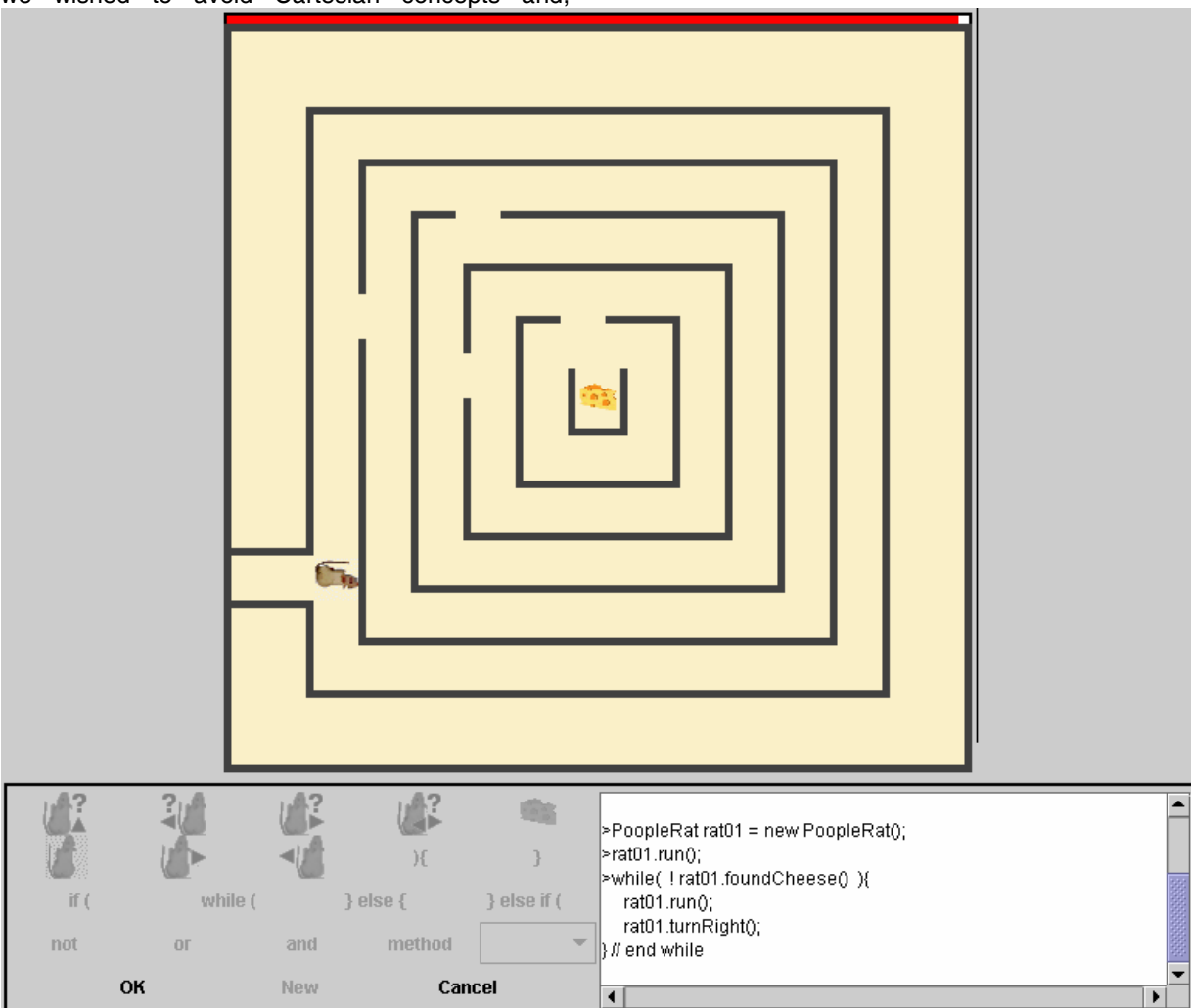


Figure 1 The poopRat POOPLE

2. THE POOPLES

The simplest POOPLE, known as poopRat, is illustrated in Figure 1. The scenario is a rat having to run through a maze and reach the cheese before it expires from hunger. The rat, an instance of the Rat class, the rat is able to respond to movement messages such as run(), turnLeft(), turnRight(). It is also able to give answers to inquiry methods such as clearForward(), clearLeft(), clearRight(), foundCheese().

These messages are sent to the rat by pressing the corresponding button on the control panel at the lower left of the interface. As the controls are interacted with, the corresponding Java code is assembled in the lower right text area, respecting the canonical format. At any stage the *Cancel* button can be pressed to abandon the current instructions. When appropriate the *OK* button is enabled allowing the instructions to be sent to the rat, where they are executed.

For example in Figure 1 pressing the *OK* button would send the current instructions, the while loop, to the rat. The consequence of which would be that the rat would run up and down the left hand side of the maze until it ran out of energy and died. The energy level being indicated by the bar at the top of the maze.

This system is highly modal and is driven by a complex underlying state transition model. This ensures that at any point in the interaction between the user and the rat the only buttons that are available are those that will produce syntactically correct Java. The intention is that problems relating to Java syntax are removed from the student's perception leaving only problems relating to semantics.

There are three implicit levels of control available. The first is direct control where the rat is driven directly by means of the movement primitives. The second is where the control structures, ifs and whiles, can be used to manipulate the rat. The third level involves declaring and defining methods which can then be used directly, or within control structures, or within other methods.

Two other POOPLES have been constructed. The poopSub is illustrated in Figure 2 and involves driving a submarine through a series of baffles before it runs out of air. The other is poopMedic, illustrated in Figure 3, this is more abstract in its visualisation and more complex in its controls. Its context involves driving an ambulance through a regular grid of streets and avenues to reach a patient before they expire.

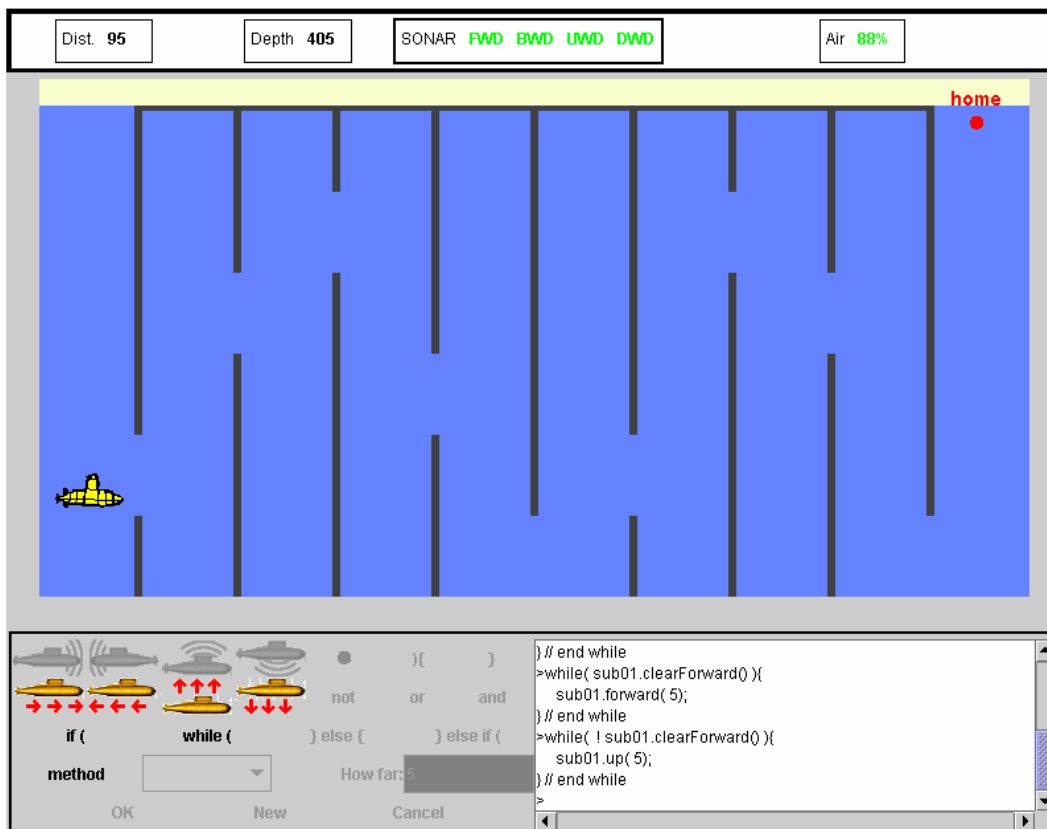


Figure 2 The poopSub

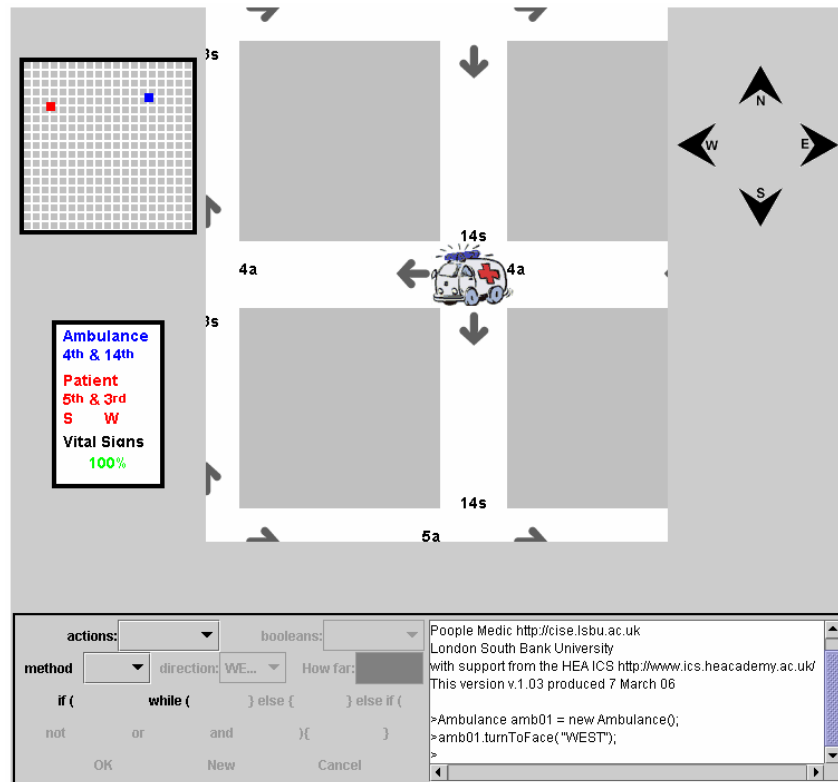


Figure 2 The poopMedic

3. EVALUATION

During the development of the POOPLES they have been shown at a number of gatherings of computing lecturers and have attracted much favourable comment. Several Universities have made use of them, mainly as contextualising material for ab-initio taught masters

Two small evaluation studies have been performed with groups of further education students aspiring to study computing or information technology at university. After being introduced to the poopRat and poopSub they were provided with guides to the use of the software and left to work with the environments for about an hour. They were then invited to fill in a short questionnaire.

A total of 32 responses were received of which 18 stated that they had never programmed before, 9 that they had middling experience and 6 that they had done lots.

Of those who had never programmed 13 (56%) stated that they thought they would enjoy programming. The remainder stated that were not sure, none said that they thought they would not enjoy programming.

Of those who had programmed a little a small number (2) indicated that they actively disliked programming and they felt that the POOPLES would not help them learn. The remainder were neutral or

positive towards programming and felt that the POOPLES might help. The group that had lots of experience were more positive than the middle group that the POOPLES would help them learn.

All respondents were asked if they had a preference between the rat and the sub. Overall there was no preference between the two POOPLES (a score of 2.57 where 2.5 indicates neutrality). Although there was an indication that the experienced group preferred the rat (2.66 cf 2.55).

In addition the students were given a free response to comment about what were the best and worst things about the POOPLES. The positive responses focussed upon the task not the syntax ("learning how to command the rat", "good insight into programming", "not having to type"). The negative comments focussed upon not having an undo capability ("... an error was made, having to restart the sub", "starting over and over ..."), and also indicated a desire for a more permanent record of the code produced.

Observationally during these, and other sessions, the students were highly engaged with the environments, were operating independently of the tutors and were very task focussed. However during the early usage sessions it was noted that most students stayed at the direct control level of command. This was addressed by the production of support material which directed the student's attention to the control structures and the use of

methods. Also observationally the POOPLES were noticeably more motivating to teenage boys than girls.

4. CURRENT STATUS & FUTURE WORK

The three POOPLES, the rat the sub and the medic have been shown to be stable and robust enough for classroom usage with students as young as 14. They are implemented as 100% Java applets and so should operate upon all platforms and operating systems that support a suitable JVM.

The evaluations have indicated that they can have a motivating effect upon students who have yet to start programming. Worryingly it also suggests that students who already dislike programming were not re-motivated by experiencing them. For those who do enjoy programming it was noticeable that the more complex sub was preferred over the simpler rat. The free response comments indicated that the intention of avoiding syntax barriers had been accomplished. However these were very small scale studies.

There are some small upgrades planned for the rat and the medic. For the rat the maze will be changed to create dead ends, requiring a more sophisticated algorithm to reach the cheese. Likewise for the medic road works and fire engines will be used to block off some roads, again requiring a more sophisticated algorithm. Both of these upgrades will be on a software switch allowing the simpler or more complex environment to be presented. The effect of this will be to give five different programming tasks to the students.

The issue of being able to undo and obtaining hard copy of the code will require additional engineering. The user interface is already cluttered and providing additional capabilities would add to its complexity.

5. ACKNOWLEDGMENTS

The initial development of the POOPLES was supported by a learning & teaching fellowship grant from the faculty of Business, Computing and Information Systems (BCIM) at London South Bank University (LSBU). Later development was supported by the Higher Education Academy subject Centre for Information & Computer Science (HEA/ICS) and LSBU. The POOPLES are freely available at the URL given at the start of this paper and at the HEA/ICS website and have also been archived in JORUM [14].

6. CONCLUSIONS

The POOPLES are stable and available as small granularity learning objects to introduce some concepts and experiences of programming to students who have little or no experience. They have a much lower 'buy in' than Alice or ToonTalk

and embody a more concrete microword and task focus than Karel or Jeroo. At a time when fewer and fewer students are choosing to study computing at university they are an additional resource that can be used to attempt to persuade them that computing can be fun.

7. REFERENCES

- [1] Culwin Fintan, Adeboye Kemi & Campbell Phil A Bridging, Scaffolding or Skeletal Initial OOSD Learning Object. Proc. 5th LTSN-ICS Conference (2004). Pages 6-10.
- [2] Bonar, J. and Soloway, E. Preprogramming knowledge: a major source of misconceptions in novice programmers, Human-Computer Interaction, Vol. 1, 1985, pp. 133-161
- [3] Elliot Soloway & James C. Spohrer Studying the Novice Programmer Lawrence Erlbaum Associates 1988, ISBN: 0805800026
- [4] Becker B W, Teaching CS1 with Karel the robot in Java, Proc. SIGCSE 32, 2001 50-54
- [5] Karel is available at <http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html> (accessed 1 April 2005)
- [6] Sanders D & Dorn B, Jeroo: a tool for introducing object-oriented programming, Proc. SIGCSE 34, 2003, 201-204
- [7] Jeroo is available at <http://www.jeroo.org> (accessed 1 April 2005)
- [8] Conway M et. al. Alice: lessons learned from building a 3D system for novices Proc. SIGHCI 2000, 486 – 493
- [9] Alice is available at <http://www.alice.org> (accessed 1 April 2005)
- [10] Ken Kahn, ToonTalk – Steps Towards Ideal Computer-Based Learning Environments, in Tokoro M Steels L, A Learning Zone of One's Own, los Press, 2004, ISBN 2004
- [11] ToonTalk is available at <http://www.toontalk.com> (accessed 1 April 2005)
- [12] Eric Allen, Robert Cartwright & Brian Stoler, DrJava: a lightweight pedagogic environment for Java, ACM SIGCSE Bulletin, Volume 34 , Issue 1 (March 2002), Pages: 137 – 141
- [13] Robert Sheehan, Incremental Control of a Children's Computing Environment, Human-Computer Interaction, Proc Interact 99, pp504-509
- [14] JORUM The JISC Online Repository for Learning and Teaching Materials, www.jorum.ac.uk/ (accessed 1 April 2005).

