Do Robots Dream of Virtual Sheep: Rediscovering the "Karel the Robot" Paradigm for the "Plug&Play Generation"

Eike Falk Anderson
eanderson@bournemouth.ac.uk

Leigh McLoughlin
lmcloughlin@bournemouth.ac.uk

The National Centre for Computer Animation
Bournemouth University, Talbot Campus
Fern Barrow, Poole, Dorset BH12 5BB, UK

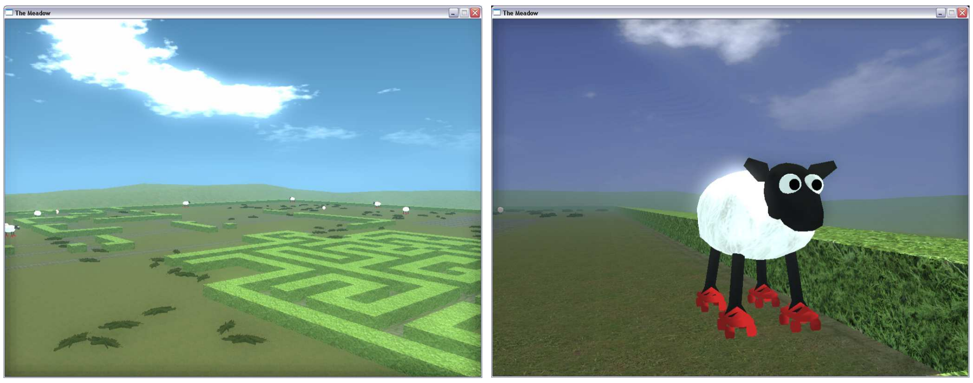

Figure 1: ”The Meadow” virtual environment.

ABSTRACT
We introduce ”C-Sheep”, an educational system designed to teach students the fundamentals of computer programming in a novel and exciting way. Recent studies suggest that computer science education is fast approaching a crisis - application numbers for degree courses in the area of computer programming are down, and potential candidates are put off the subject which they do not fully understand. We address this problem with our system by providing the visually rich virtual environment of ”The Meadow”, where the user writes programs to control the behaviour of a sheep using our ”C-Sheep” programming language. This combination of the ”Karel the Robot” paradigm with modern 3D computer graphics techniques, more commonly found in computer games, aims to help students to realise that computer programming can be an enjoyable and rewarding experience and intends to help educators with the teaching of computer science fundamentals. Our mini-language-like system for computer science education uses a state of the art rendering engine offering features more commonly found in entertainment systems. The scope of the mini-language is designed to fit in with the curriculum for the first term of an introductory computer programming course (using the C programming language).

In Proceedings of Game Design and Technology Workshop and Conference 2006, Liverpool, UK, pages 92-95.

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

General Terms
Human Factors, Languages

Keywords
Pedagogy, Programming, Visualisation, Games

1. INTRODUCTION
In recent years there has been a major shift in computer science curricula with the adoption of the ”objects first” approach (emphasizing object-oriented design and programming). This has subsequently led to a redesign of many introductory computer programming courses. New studies suggest that computer science education is fast approaching a crisis as enrolment numbers to degree courses have fallen to extremely low levels [7]. One of the reasons why that may be is a general misconception of computer science and programming among prospective students. A lack of knowledge of the subject area may well be the underlying reason for this bias against computer science. Computer science itself as well as related subjects are wrongly perceived as boring (non-creative), unglamorous and difficult (i.e. actually requiring work) and computer programming especially is often regarded as a monotonous and uninteresting task. We have encountered prospective students

who appear intimidated by the technological aspects of the course we offer, as well as the maths involved. Other students are over-confident after having taken an information technology course (little more than a computer literacy course) at school, only to quickly lose interest once they have embarked on the computer science related course (or module) as soon as they realise their mistake, an observation also made by Beaubouef and Mason [2]. While this misconception might in part explain the drop in student numbers, the fact that educators now also debate ”whether or not the objects early approach has failed” [19] might indicate that a return to the older ”procedures first” method of computer science education might be called for. It has been suggested that one way to achieve a greater uptake of computer science would be to make programming ”more fun” [7]. This is much easier said than done, as the challenge that educators face when teaching computer science in general and computer programming in particular is to maintain the interest of students in the subject. A major part of this problem appears to be a lack of patience that we have observed among students. The students from the ”Plug&Play generation” expect to see immediate (and spectacular) results, often before they have learned enough to achieve anything remotely spectacular. An analogy for this would be an illiterate person setting out to write a bestselling novel. A side effect of this clash of realities is that students' programs often have unintended results, causing confusion and once faced with difficulties (a recent study of which was made by Lahtinen et al [13]), their motivation suffers and they quickly lose interest in the subject. In our experience the result of this is that the students fall behind in their studies which causes frustration and a further loss of motivation, ending in a downwards spiral and eventual exam failure. Motivation is a major factor in the success of students of programming who need to practice writing programs to improve their skills [10]. The fact that students have taken up a course involving programming is alone no indicator for the existence of motivation. This is something we have observed among students of our course (Computer Animation and Visualisation), an arts degree with a strong technical component. Our students come from very diverse backgrounds, often with little prior experience with computers. As a result we especially face motivational problems among those students who joined the course out of interest for its artistic elements, who consider the computing aspects of the course as a necessary evil. We intend to address these problems with our C-Sheep system (see figure 2), a re-imagination of the ”Karel the Robot” paradigm using modern 3D computer game graphics that today's students are familiar with, our aim being to motivate students to take up programming and to provide them with an enjoyable experience at the same time.

2. TEACHING WITH ROBOTS IN VIRTUAL WORLDS

It is generally understood that programming cannot be learned from reading books on the subject alone, but only by practicing it by actually writing programs on a computer. The question that therefore needs to be answered is: ”how can students be motivated to practice programming?”

A suitable answer to this question is provided by the ”Karel the Robot” paradigm which has been reported to have proven itself as highly successful [12, 20].

2.1 The "Karel the Robot" Paradigm

The ”Karel the Robot” paradigm consists of the use of a mini-language [5], that provides a small number of instructions and which allow users to take control of virtual entities, acting within a micro world. It is named after the very successful ”Karel the Robot” program [17], one of the widest known computer science teaching tools which uses a program structure based on the syntax of the Pascal programming language. Untch describes Karel as ”essentially a programmable cursor that can move across the flat world” of a 2D grid with obstacles (walls) that cannot be passed and objects (beepers) that can be placed in or removed from the micro world [21]. The instructions found in mini-languages are usually a set of actions to be taken by the virtual entity in the virtual environment, as well as a set of (sensor) queries, providing information about the immediate surroundings of the virtual entity in the micro world it inhabits.

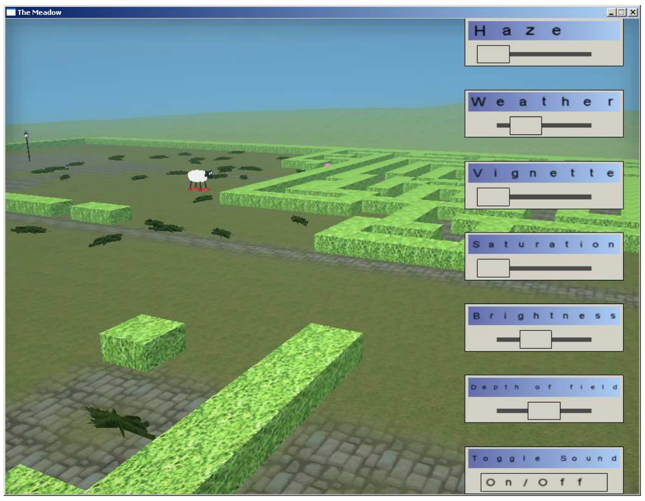

Figure 2: ”The Meadow” and the C-Sheep UI (running on a low-spec graphics card)

The success of the ”Karel the Robot” paradigm is based on a number of factors:

- By providing a game-like setting for the task of computer programming, the students' imagination is captured and their interest is maintained. Students are motivated to spend more time programming and are rewarded with an enjoyable experience. In our experience, this heightened motivation is likely to have the side effect that fewer students question the relevance of tasks they have been set, a problem often faced by educators if students are confronted with toy-problems in programming exercises.

- The graphical representation of the micro world provides instant visualisation of the algorithms used in the programs controlling the virtual entities - their position and orientation within the virtual world show the current state of the program. This is especially useful as many problems faced by novice programmers can possibly be traced back to an inadequate understanding of program state [9]. The visual feedback is invaluable to the understanding of how a given algorithm works, where in the program potential errors occur and, consequently, how these errors can be debugged.

- Data required by programs is less complex than would be in a real-world programming language. This is achieved by scaling down the data representation to the visual program state - the mini-language uses minimal syntax and is kept variable free to provide an environment with minimal complexity, making this program state ”more real to ... students than collections of alphanumeric values stored in aliased memory locations” [6].

2.2 Shortcomings of ’Traditional’ Mini-Languages

Despite the pedagogical benefits described above, the traditional mini-languages have several shortcomings. While existing systems may again be relevant, we believe that they are now severely outdated. Whereas two decades ago, students would be intrigued by the 2D top-down representation of the micro world (often restricted to ASCII characters in text-mode), we are convinced that it is extremely difficult - not to say impossible - to maintain the interest of students from today’s ”Plug&Play generation” by using a text-based graphical representation. This assumption is reinforced by the negative student reaction to the 2D top-down representation of the Robocode system [14], reported by Bierre et al [3]. To prevent mini-language based teaching from becoming obsolete, existing systems cannot just be recycled but must be updated. While traditional mini-language systems have employed a strictly 2D top-down representation, more recent offerings also use pseudo 3D graphics (using isometric projection). Surprisingly few examples, such as Alice or the MUPPETS system [8, 18], use true 3D graphics which are ”attractive and highly motivating to today’s generation of media-conscious students” [16] for their micro world representation. It is this use of the visual gimmickry of modern computer games for representing the virtual world, however, which is most likely to help with meeting the high expectations of the ”Plug&Play generation”.

A different problem is often the choice of programming language on which the mini-language is based. Real-world programming languages often require a lot of work and understanding before meaningful results are obtained, which tends to frustrate impatient students. On the other hand, very abstract ’toy languages’ that could be used as an alternative and which offer immediate results but require only little understanding are often too far removed from real-world systems to appear relevant. We believe that the solution in this case must be a sort of ’toy language’ which is not purely a learning instrument but both simple and close to real-world systems.

We aim to overcome these problems with our C-Sheep system: The C-Sheep programming language, ”The Meadow” virtual environment and the C-Sheep library.

3. C-SHEEP

C-Sheep replaces robots with sheep and places them into a 3D computer-game-like virtual environment instead of a simple 2D grid. As the course that C-Sheep was designed for uses the C programming language [11], the C-Sheep language has been designed as a subset of ANSI C. Within the confines of this C subset, C-Sheep implements the control structures that are required for teaching the basic computer science principles encountered in structured programming [4], these being the (unconditional) sequence, conditional statements and loops. In terms of C, the control structures available in C-Sheep are the block, if and if-else alternatives, as well as while and do-while loops. C-Sheep supports the declaration and use of variables. While this might be considered as a potential problem by some since it adds a further level of complexity to the mini-language, others have observed that a lack of variables will lead to problems in a transition to real-world languages [21]. C-Sheep allows the use of variables for arithmetic expressions which may be useful to track object’s histories in the virtual environment (as non-visual states), which has been recognised as possibly beneficial [9]. Like C, C-Sheep has mechanisms for the definition of sub-routines which may be called recursively. Pre-defined actions (instructions for controlling sheep entities in ”The Meadow” virtual environment) and (sensor) queries that can be performed by the sheep entity are disguised as library functions: To introduce novice programmers to the concept of code modularisation and libraries, C-Sheep programs must contain an include statement to (supposedly) parse the function prototypes in order to access library functions, while internally these functions are actually intrinsic to the virtual machine. Some of these sheep-specific instructions allow the querying of states in the virtual world (e.g. the current state of the weather - see figure 3). These world states can be altered interactively by the user (while C-Sheep programs are running), adding a separate layer of interactivity to the learning game. By instigating a state change in the virtual environment, the user can cause different sections of C-Sheep programs to be executed, allowing experimentation with different behaviours of the sheep entity from within the same C-Sheep program. Dann et al state that this use of 3D computer graphics to represent program state is ”intrinsic in the natural way to view the data itself” [9].

The C-Sheep program controlled entities exist in the 3D virtual environment of ”The Meadow” virtual world. This 3D game-like representation is essential to interest students from the ”Plug&Play generation” in computer programming. ”The Meadow” is based on our proprietary ”Crossbow” game engine [15]. The Crossbow Engine is a compact game engine which is flexible in design and offers a number of features common to more complex engines. Designed specifically for ”The Meadow”, it incorporates a robust virtual machine for executing C-Sheep programs which is based on the ZBL/0 virtual machine [1].

The C-Sheep system also includes a counterpart library (in the current stage of development only visualising C-Sheep programs in a 2D top-down view) for programs written in the C programming language. The functions provided by this library mirror the C-Sheep instructions for the virtual entities. As suggested by Untch, the purpose of such a library is to simplify the migration from the educational mini-language to real-world systems [21], in our case from C-Sheep to the C programming language. Using the library, C-Sheep programs can be compiled into an executable using a normal off-the-shelf C/C++ compiler, allowing novice programmers to make an easy transition from using the C-Sheep system to C. The compiled executable can then be run from within the native working environment of the operating system.

4. SUMMARY AND FUTURE WORK

Computer programming is an essential skill for software developers and as such is always an integral part of every computer science curriculum. However, even if students are pursuing a computer science related degree, it can be very difficult to interest them in the act of computer programming, the writing of software, itself. To address this problem we have presented C-Sheep, an educational system for the teaching of computer science principles and a tool for learning the basics of the C programming language. It contains only a small number of reserved words from the C programming language, acting as a mini-language subset. C-Sheep follows in

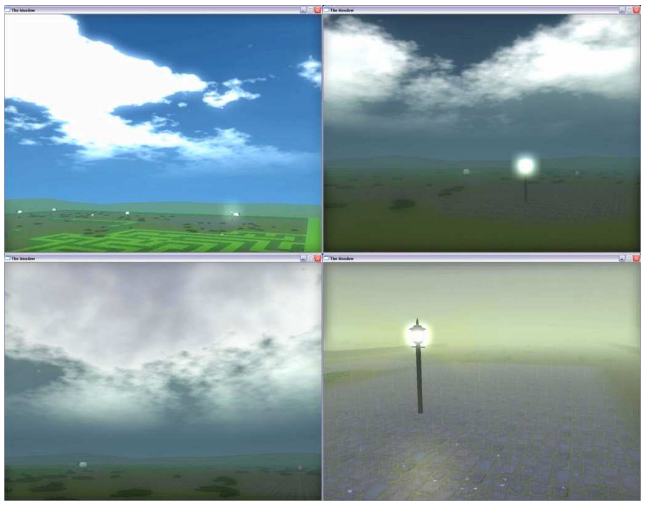

Figure 3: Weather in ”The Meadow”

the tradition of mini-languages, started by ”Karel the Robot”, but employs a state-of the art games rendering engine for algorithm visualisation. C-Sheep consists of a task-specific set of instructions and queries, which allow users to control virtual entities (in the case of C-sheep, sheep) within ”The Meadow” 3D virtual environment, the micro world which they inhabit. C-Sheep uses a real-world language (ANSI C) as its basis and also provides a counterpart library for easy migration from the C-Sheep system to real world compilers. This is where C-Sheep differs from other educational systems as even those that use a more C/C++ or Java-like system are often severely limited in the syntax they use - mainly because their (often variable-free) design explicitly tries to remove language complexity.

There are several directions that we are planning to explore in future versions of the C-Sheep system. Greater interactivity (including communication among several sheep), additional in-game objects and entities (like a sheep-dog) and an integrated development environment with a JIT (just-in-time) compiler are possible extensions to the system. The current prototype uses colour-coded bitmap-images to store game levels and C-Sheep program source code is written in a normal text editor, so better - possibly integrated - authoring tools for creating different scenarios would be a highly desirable improvement. Further refinement of the system would be the provision of more comprehensive documentation (in addition to the current language specification). Finally, feedback from using C-Sheep in the classroom will hopefully allow us to fine-tune the system. The C-Sheep system itself has not yet been used for teaching. So far students have only been exposed to a problem solving and algorithm design exercise (on the white-board) using the C-Sheep language. This exercise was very well received by the students which leads us to believe that our system is suitable for the task it was designed for, but conclusive proof will only be available after we have collected data from a trial run of the C-sheep system. For this we are planning to introduce the current prototype of the C-Sheep system as a teaching tool for the first term of the first year computer programming unit at the National Centre for Computer Animation (Bournemouth University).

Further information on the C-Sheep system is available at
http://ncca.bmth.ac.uk/eanderson/C-Sheep/

5. ACKNOWLEDGEMENTS

First and foremost we would like to express our gratitude towards our supervisor, Prof. Peter Comninos. Without his support this project would not have been possible. We would also like to thank our colleagues, especially Olusola Aina, for their comments and suggestions that have contributed to this project. Finally we need to mention Dominic Halford. It is his ”fault” that our programs control sheep instead of other animals.

6. REFERENCES

[1] E. F. Anderson. A npc behaviour definition system for use by programmers and designers. In *Proceedings of CGAIDE 2004*, pages 203–207, 2004.

[2] T. Beaubouef and J. Mason. Why the high attrition rate for computer science students: some thoughts and observations. *ACM SIGCSE Bulletin*, 37(2):103–106, 2005.

[3] K. Bierre, P. Ventura, A. Phelps, and C. Egert. Motivating oop by blowing things up: an exercise in cooperation and competition in an introductory java programming course. In *SIGCSE ’06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 354–358, 2006.

[4] C. Böhm and G. Jacopini. Flow diagrams, turing machines and languages with only two formation rules. *Communications of the ACM*, 9(5):366–371, 1966.

[5] P. Brusilovsky, E. Calabrese, J. Hvorecky, A. Kouchnirenko, and P. Miller. Mini-languages: A way to learn programming principles. *Education and Information Technologies*, 2(1):65–83, 1997.

[6] D. Buck and D. J. Stucki. Jkarelrobot: a case study in supporting levels of cognitive development in the computer science curriculum. In *SIGCSE ’01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, pages 16–20, 2001.

[7] L. Carter. Why students with an apparent aptitude for computer science don’t choose to major in computer science. *ACM SIGCSE Bulletin*, 38(1):27–31, 2006.

[8] S. Cooper, W. Dann, and R. Pausch. Alice: A 3-d tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, 15(5):107–116, 2000.

[9] W. Dann, S. Cooper, and R. Pausch. Making the connection: Programming with animated small world. In *Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*, pages 41–44, 2000.

[10] T. Jenkins. The motivation of students of programming. In *ITiCSE ’01: Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 53–56, 2001.

[11] B. W. Kerninghan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, 1988.

[12] K. L. Krause, R. E. Sampsell, and S. L. Grier. Computer

science in the air force academy core curriculum. In *SIGCSE '82: Proceedings of the thirteenth SIGCSE technical symposium on Computer science education*, pages 144–146, 1982.

[13] E. Lahtinen, K. Ala-Mutka, and H. J&#228;rvinen. A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*, 37(3):14–18, 2005.

[14] S. Li. Rock 'em, sock 'em robocode! IBM developerWorks: Java technology - http://www-106.ibm.com/developerworks/library/j-robocode/, 2002.

[15] L. McLoughlin and E. F. Anderson. I see sheep: A practical application of game rendering techniques for computer science education. Poster at Future Play '06 Conference, 2006.

[16] B. Moskal, D. Lurie, and S. Cooper. Evaluating the effectiveness of a new instructional approach. *ACM SIGCSE Bulletin*, 36(1):75–79, 2004.

[17] R. E. Pattis. *Karel the Robot, a Gentle Introduction to the Art of Programming*. John Wiley and Sons, 1981.

[18] A. M. Phelps, B. K. J., and D. M. Parks. Muppets: multi-user programming pedagogy for enhancing traditional study. In *CITC4 '03: Proceedings of the 4th conference on Information technology curriculum*, pages 100–105, 2003.

[19] S. Reges. Back to basics in cs1 and cs2. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 293–297, 2006.

[20] E. Roberts. Strategies for encouraging individual achievement in introductory computer science courses. In *SIGCSE '00: Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, pages 295–299, 2000.

[21] R. H. Untch. Teaching programming using the karel the robot paradigm realized with a conventional language. On-line at: http://www.mtsu.edu/˜untch/karel/karel90.pdf, 1990.