C-Sheep: Controlling Entities in a 3D Virtual World as a Tool for Computer Science Education

Eike Falk Anderson and Leigh McLoughlin
The National Centre for Computer Animation
Bournemouth University, Talbot Campus
Fern Barrow, Poole, Dorset BH12 5BB, UK

ABSTRACT

One of the challenges in teaching computer science in general and computer programming in particular is to maintain the interest of students, who often perceive the subject as difficult and tedious. To this end, we introduce C-Sheep, a mini-language-like system for computer science education, using a state of the art rendering engine, usually found in entertainment systems. The intention is to motivate students to spend more time programming, which can be achieved by providing an enjoyable experience. In the C-Sheep system this is aided by the visual gymnastics of modern computer games, which allows programs to provide instant visualisation of algorithms. This visual feedback is invaluable to the understanding of how the algorithm works, and if there are unintended results - how errors in the program can bedebugged.

The C-Sheep System

Figure 1: Sheep entity on “The Meadow”

Introduction and Problem

Recent studies suggest that computer science education is fast approaching a crisis - enrolment numbers to degrees courses have fallen to extremely low levels [6]. One of the reasons for this may be 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 the lack against computer science. Computer science and related subjects are wrongly conceived as difficult and boring (non-creative) and computer programming as repetitive and uninteresting task. Computer programming is an essential skill for software developers and is such as an integral part of every computer science curriculum. However 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. It is generally understood that programming cannot be learned from reading books on the subject but only by practicing it, yet therein lies the problem: “How can students be motivated to practice programming?”

Rationale for Educational Programming Languages

It has been suggested that one way to achieve a greater uptake of computer science would be to make programming “fun” [6]. Using a mini-language [3] as the introductory programming language readily fulfills this requirement. These mini-languages usually provide a task-specific set of instructions (and error) queries which allow users to take control of virtual entities, acting within a micro world (see figure 1). This micro world provides a graphical representation of the algorithms used in the programs controlling the virtual entities. Their position and orientation within the virtual world visualise the current state of the program which is especially useful as many problems faced by novice programmers can possibly be traced back to an inadequate understanding of program state [8].

The mini-language system “Karel the Robot” for example [9] (using a “toy-language” based on the syntax of the Pascal programming language) is one of the best known computer science teaching tools and has had considerable success. The aim for all of these systems is to motivate students to take up programming and to provide them with an enjoyable experience at the same time. This is achieved by providing a game-like setting for the task of computer programming. Until now, most mini-language systems used in teaching have employed a strictly 2D top-down representation, some use pseudo-3D graphics (using isometric projection) with - surprisingly - only a few examples, such as Alice [5], using true 3D graphics. Dann et al. state that a 3D representation of the program state is “essential in the natural way to view the data itself” [9]. It is therefore especially important to aim for a 3D game-like representation in mini-languages to interest the “PlayFlag generation” in computer programming.

C-Sheep Programming Language

The C-Sheep programming language is a subset of the ANSI C programming language [7] (see figure 2). Apart from just being a tool for teaching the basics of the C programming language, C-Sheep implements the C control structures that are required for teaching the basic computer science principles encountered in structured programming. These control structures are the (unconditioned) sequence, conditional statements and loops [2]. C-Sheep also allows the definition of sub-routines (functions) which can be called recursively. Unlike other programming languages which have minimal syntax and which are variable free to provide an environment with minimal complexity, C-Sheep allows the declarative and use of variables. To access library functions, C-Sheep programs must include header files (supposed) containing function prototypes. This is done only to introduce novice programmers to the concept of code modularisation and libraries, while internally the C-Sheep library functions are actually intrinsic to the system. The C-Sheep standard libraries (see table 1) provide a number of general purpose functions, as well as functions for controlling sheep entities in the virtual environment. Some of the latter allow the opening of changes in the virtual world (e.g. the current state of the weather - see figure 3). These changes can be registered interactively by the user (while programs are running).

To execute native C-Sheep programs they must be compiled with the C-Sheep compiler which translates from the C-Sheep subset of the C programming language to virtual machine bytecodes.

C-Sheep Standards Libraries

<table>
<thead>
<tr>
<th>Standard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdlib.h</td>
<td>(subset of ANSI C stdlib.h)</td>
</tr>
<tr>
<td>assert</td>
<td>assert failure</td>
</tr>
<tr>
<td>math.h</td>
<td>(subset of ANSI C math.h)</td>
</tr>
<tr>
<td>sheep</td>
<td>sheep entity control functions</td>
</tr>
</tbody>
</table>

Table 1: Functions of the C-Sheep standard libraries

C-Sheep Syntax

![Syntax Diagrams for the C-Sheep Programming Language](image)

C-Sheep Standard Libraries

<table>
<thead>
<tr>
<th>preprocessor-directive</th>
<th>declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>#include</td>
<td>Ident</td>
</tr>
<tr>
<td>void</td>
<td>Ident</td>
</tr>
<tr>
<td>int</td>
<td>Ident</td>
</tr>
<tr>
<td>float</td>
<td>Ident</td>
</tr>
</tbody>
</table>

C-Sheep Syntax

“The Meadow” virtual environment in the virtual world in which entities are controlled by C-Sheep programs exist. It is based on our proprietary “Crossbow” game engine which incorporates a virtual machine for executing C-Sheep programs.

The Crossbow Engine is a compact game engine designed specifically for “The Meadow”, yet it is flexible in design and offers a number of features common to more complex engines [3].

The Crossbow Virtual Machine is a module of the Crossbow Engine. It is an improvement on the ZBL/0 virtual machine [1] and includes the use of pointers as well as facilities for the creation of aggregate data types (arrays and record structures). The virtual machine is targetable to different languages, i.e. a Java based C-Sheep or Pascal based P-Sheep could be created with relatively little effort.

The Crossbow Virtual Machine is a module of the Crossbow Engine. It is an improvement on the ZBL/0 virtual machine [1] and includes the use of pointers as well as facilities for the creation of aggregate data types (arrays and record structures). The virtual machine is targetable to different languages, i.e. a Java based C-Sheep or Pascal based P-Sheep could be created with relatively little effort.

Sample C-Sheep Program

```c
/* C-Sheep Sample Program */

#include "CSheep.h"

int main()
{
    // Start a run in "The Meadow"
    // The program when the original position has been reached
    printf("Welcome!
");
    return 0;
}
```

A sheep running this program will traverse a maze, looking for a ball. Once a ball is found, the sheep will be blocked by the ball and will only be able to move in the direction that will lead it to the next correct move (randomly) with a small chance of backtracking and a smaller chance of randomly changing direction. The starting point for the maze is the same after the algorithm has been run.

Summary and Future Work

We have presented C-Sheep, a system for the teaching of computer science principles using a subset of the C programming language within a 3D computer game-like virtual environment. We 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 the we are currently planning to introduce 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).

Acknowledgements

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

No Sheep were harmed during the creation of this poster.

References


Lucas the Sheep design by Alexandre Nakonechnyj (nako@openlink.com.br).