Game Programming in Introductory Courses With Direct State Manipulation

Michael Kölling
Computing Laboratory
University of Kent
mik@kent.ac.uk

Poul Henriksen
Computing Laboratory
University of Kent
p.henriksen@kent.ac.uk

ABSTRACT

While the introduction of object-oriented programming slowly moves down the age groups – starting from advanced university courses, to introductory courses, and now into high schools – many attempts are being made to make object-oriented programming introduction less abstract and theoretical. Visualisation and interaction techniques are being applied in an attempt to give students engaging and concrete experiences with objects. Recently, the *greenfoot* environment has been proposed as another step in this development. In this paper, we describe new functionality in the greenfoot environment, especially the addition of user interaction programming via direct state manipulation. Direct state manipulation provides very low overhead graphical I/O handling at a level that makes it feasible to guide students to simple graphical game programming within a few weeks, while concentrating on fundamental object-oriented concepts in the structure of the program.

Categories and Subject Descriptors

K.3.2 [**Computers & Education**]: Computer & Information Science Education - *Computer Science Education*

D.1.5 [**Programming Techniques**]: Object-Oriented Programming.

General Terms

Design, Human Factors

Keywords

Pedagogy, Object-Oriented Programming, Visualisation, Animation, Behaviour, Interaction, Games

1. INTRODUCTION

A few years ago, authors writing about the introduction of object-oriented programming to students argued for an 'objects early' approach. Object orientation should be moved from advanced programming courses in later years of the curriculum to first programming courses in order to avoid the paradigm shift when moving into object orientation. This change was largely accepted and implemented in most institutions over the last years.

The 'objects early' debate then continued, this time arguing that the concept of objects and classes should be addressed in the early weeks of the introductory course, not towards its end. Several pedagogical reasons were stated, claiming better understanding of important concepts with this approach. We do not want to continue that particular debate here, but merely point out that we agree with the arguments of the objects-early proponents.

ITiCSE'05, June 27–29, 2005, Monte de Caparica, Portugal.

Copyright 2005 ACM 1-59593-024-8/05/0006…$5.00.

One more recent development in this debate is that the goal posts are shifting. While we aim at introducing object orientation first, for many students the introductory programming course at university or college is not the first contact with programming anymore. Programming is now regularly taught at high school level, and it is possible that its introduction will move even further down into mid-level schools. If appropriate tools were available, this certainly seems possible.

For teachers of object orientation this introduces a radical change. If we want to teach objects early, we can no longer concentrate on college courses, we have to address students at school level.

There are several significant differences between those two populations (college versus school students) that are highly relevant for teachers as well as for developers of pedagogical content and tools. Apart from maturity issues, the most significant difference is interest.

In many computing courses at university and college level, students have made a conscious choice to study computing, and an individual interest (or at least some form of secondary motivation) can be assumed in a substantial part of the student audience. (Those colleges where this is not true need to be viewed as being similar to high schools for the purpose of this discussion.)

At high schools, this is not the case. Many students have no interest in programming, either because they do not know anything about it, or because they dislike the idea of programming based either on prior experience or prejudice.

Thus, in a school setting, an introduction to programming must address distinctly different challenges than a similar course at a university. The course must not only convey programming concepts, it must first and foremost generate interest in the subject matter for students with no previous affinity for the subject.

In this paper, we introduce a tool named *greenfoot*, which is designed to form the basis of an introduction to programming for school students or at early college level.

Greenfoot is an interactive object world that aims at motivating students by providing concrete experience with object concepts through interaction and visualisation, using engaging context scenarios, while conveying important object-oriented programming abstractions in the standard Java programming language.

We first discuss some of the more fundamental considerations in designing such a tool, followed by a description of the greenfoot interaction and visualisation capabilities. Specifically, we introduce direct state manipulation as a novel mechanism for easily programming interactive systems.

2. MAKING OBJECTS CONCRETE

In order to engage the interest of young students, we aim at providing concrete experiences with the subject matter.

Since the concept of objects is at the heart of our subject matter, we aim at providing concrete experiences with objects.

Teaching about programming is in constant danger of relying heavily on abstract conceptualisation, while providing little concrete experience with the treated concepts, such as object behaviour.

In order to understand object-oriented programming students must understand objects. Programming is the activity of defining the behaviour of objects.

One of the problems in traditional programming environments is that object behaviour is not directly observable. Students can program behaviour, but only secondary effects of this behaviour can be observed (if at all), introducing an abstract separation of cause and effect.

Even in educational visualisation environments such as BlueJ [7], where objects are graphically represented, this remains true. Objects in BlueJ have a uniform appearance, and they do not change as the object acts or its state changes. Behaviour cannot directly be observed.

One educational approach that has addressed this problem is the use of micro worlds, such as turtle graphics [3], Karel the Robot [1], or the Marine Biology Case Study [9]. Dann *et al.* have used the Alice environment to address these issues using a three dimensional world [4].

In these micro worlds behaviour of objects in the world is visualised, and students can make direct observations of object behaviour and interactions.

While this is an important and valuable first step in the right direction, this approach can be taken further. Existing micro worlds suffer mainly from two restrictions: They lock students and teachers into a fixed world scenario, and they lack direct interaction mechanisms with objects or simulated worlds.

Students need to be engaged. To achieve this it is beneficial if they can interact with their artefacts. The activity of programming is one form of interaction that is at the core of these systems. But missing out on direct interaction with instantiated objects or executing simulations means missing a great opportunity for engaging especially the less technically minded students. In other words: students must be able to manipulate, experiment with, and observe objects, not merely lines of source code.

We will discuss both aspects – flexible scenarios and interaction – in more detail below. In this paper, we will concentrate especially on the interaction aspect of the greenfoot object world.

3. DESIGN GOALS

Before discussing some aspects of greenfoot in detail, we give a brief summary of the main design goals, which will help to put into perspective those decisions we discuss later in more detail.

Design goals for greenfoot include:

- to provide visual feedback of object state and behaviour;
- to allow active interaction and experimentation with object instances and to explore behaviour interactively;
- to support highly flexible scenarios, while freeing scenario writers from dealing with GUI programming;
- to provide a clean illustration of object-oriented concepts;
- to allow for easy development of interactive scenarios, for example interactive games, by students;
- to support migration to other environments.

Some of these design goals have been discussed in more detail in [5]. We will not repeat this discussion here.

The goal of supporting an interaction mechanism for running applications – for the purpose of developing game-like applications – has recently been added and was not included in earlier discussions of design goals. We describe the motivation and environment mechanism for supporting this below.

4. THE GREENFOOT SYSTEM

The greenfoot system is an interactive object world. It provides a framework and environment to create interactive, simulation-like applications in a two-dimensional plane.

One aspect of greenfoot is that it allows visualisation of appearance and location of simulation objects in a two-dimensional grid, similar to micro world systems, such as Karel J. Robot [1] or the Marine Biology case study.

In addition to this, greenfoot allows direct interactive method calls on simulation objects, similar to the interaction facilities in the BlueJ environment.

Scenarios are completely decoupled from the visualisation and interaction framework, so that greenfoot can be used for a wide variety of graphical applications.

The greenfoot system also provides a full IDE, including integrated editing, compilation, creation of new classes, object inspection and a source level debugger.

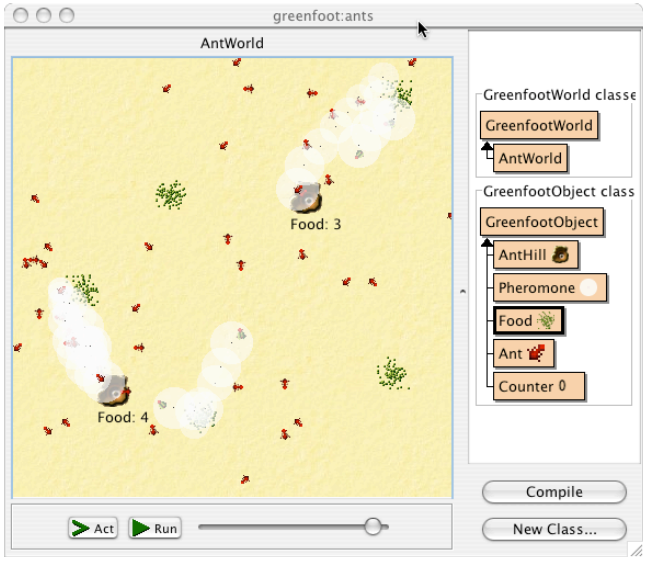

Figure 1: The greenfoot main window

4.1 The User Interface

The largest part of greenfoot's user interface is reserved for the display of the world, shown in the centre of the screen (Figure 1). It holds the greenfoot objects (ant hills, ants and food in this example).

To the right of the world is a class display. Here, all classes involved in the current application are shown. The classes are divided into *Greenfoot-World Classes*, representing worlds, and *Greenfoot-Object Classes*, representing visible objects within the world.

The classes can be edited, compiled and instantiated. These actions can be accessed from a popup menu of the class.

The lower part of the window holds execution controls to run, stop or single-step the simulation and a slider to control the execution speed.

4.2 Greenfoot Development

All classes whose instances should be visible in the greenfoot world extend the predefined superclass *GreenfootObject*. The environment also provides a predefined class *GreenfootWorld*, which implements the world itself.

The world provides a grid of cells, which can hold greenfoot objects. Each greenfoot object can specify its own individual appearance using an icon or a drawing method. Greenfoot objects have a location in the world and a rotation that is applied to the icon. The appearance can span one or more cells.

All objects in a greenfoot world are automatically animated and interactive. They can have behaviour that is exhibited when the simulation is run using the *Run* button, and they can be used for direct interaction through associated popup menus when the simulation is paused.

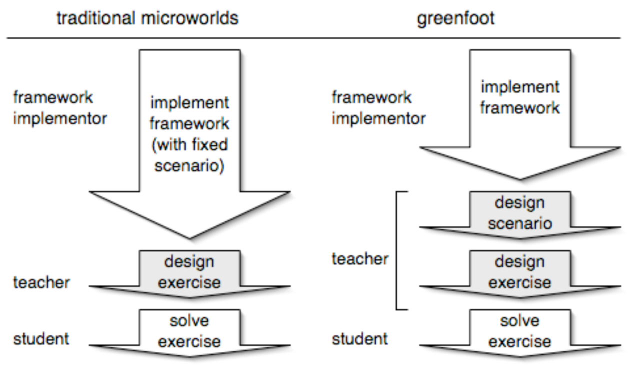

Figure 2: Roles of people involved in creation and use of micro worlds (traditional vs. greenfoot)

5. FLEXIBLE SCENARIOS

One of the important characteristics of greenfoot is the decoupling of the user level scenario from the animation and interaction framework.

Many existing micro worlds achieve simplicity by restricting use to a single scenario: The Marine Biology Case Study deals with fish and nothing else, Karel has robots and "beepers", turtle graphics has a turtle and a pen.

While this restriction has the advantage to simplify start-up, it has disadvantages as well. The scenario cannot easily be adapted for different user groups. If, for example, some students have no interest in robots, they still cannot escape them if Karel is used. It also means that courses typically use only a single scenario. The overhead of learning to use a different micro world system in order to use a different scenario is usually forbidding.

In greenfoot, a goal is to allow widely differing scenarios to be developed by knowledgeable users (such as teachers) within a single framework (Figure 2). This enables use of more user-targeted scenarios, since scenario writing is at a level of complexity that puts it within reach of many teachers. Also, it allows use of multiple scenarios in a single course, since the overhead of installing and learning to interact with a new framework is avoided.

We still envisage scenarios to be shared between teachers, but the group of people writing scenarios can easily be much larger than those who have the time to implement a complete micro world framework.

6. SOME SAMPLE APPLICATIONS

6.1 Ants

A first scenario is shown in Figure 1. This example is called "Ants" and displays ant hills, ants, food sources and pheromones. Ants leave their ant hills to find food and place tracks of pheromones if their search is successful. More food can be dropped into a running animation to influence the ants' behaviour.

During the testing phase, we have placed some ants and just a single drop of pheromones (which evaporates over time) into the world to check if ants correctly follow the pheromone smell. No additional coding is needed to perform such tests.

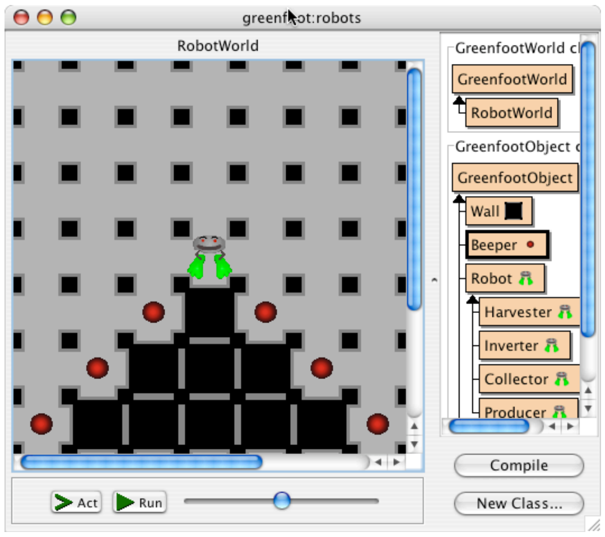

Figure 3: Karel The Robot in greenfoot

Karel The Robot

Figure 3 shows a re-implementation of the popular Karel The Robot scenario in greenfoot (with our own graphics). Robots can walk around the world and collect or place 'beepers'. Here, the world uses a grid resolution of about 30 pixels per grid cell.

In scenarios such as this one, we envisage that a teacher would create the initial robot class, while students start with making modification to the robot, and then define their own robot subclasses with specified behaviour.

The appearance of a robot, for example, can be changed with a single line of code, and additional behaviour can be added easily.

The Marine Biology Case Study is structurally very similar, and can easily be programmed as well.

6.2 A Lift Simulation

The lift simulation is a more advanced example that students might work on later in a course. We have included it here to demonstrate that greenfoot cannot only be used to display birdseye views of a regular grid surface, but also other animated two-dimensional graphics.

Here, people appear on various floors of a multi-story building, wait for a lift, and then enter the lift to travel to different floors.

Any application that uses two dimensional graphics to perform its I/O can easily be coded in greenfoot.

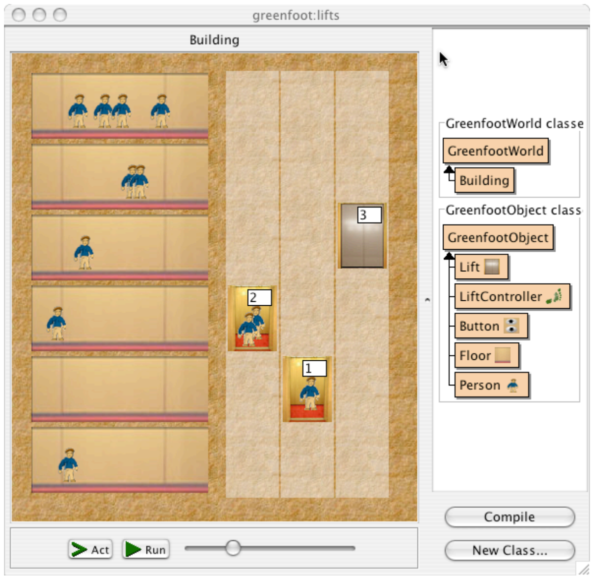

Figure 4: A lift simulation

6.3 Other scenarios

A wide range of other applications can be fitted into the greenfoot framework. While best suited to applications that produce two-dimensional graphical output, other uses are not excluded. Since drawing capabilities on the world include the drawing of text, some objects could display a behaviour that displays textual information on the screen. While this is not the main goal for greenfoot, it extends its capabilities.

Possible scenarios are unlimited. Obvious choices include emergency evacuation of buildings, traffic simulations, supermarket checkout queues, predator/prey simulations and many more. Greenfoot may even be used to provide an easy-to-use output mechanism to more advanced exercises such as, for example, the dining philosophers problem.

7. INTERACTION

The description so far has concentrated on visualisation of a continuous event simulation scenario, and interaction via method calls to selected objects while the simulation was paused.

An additional challenge was to add game-like interaction to running applications, so that students are not restricted to passive observation once a simulation has started, but can enter into an interaction with the running program.

Programming user interaction in modern object-oriented languages is often not trivial, and graphical interaction libraries are often big and complex. The most common solution for interaction in these systems is based on event-driven models.

Greenfoot uses standard Java as the user's implementation language, and the standard Java library for this purpose (included in the AWT and Swing packages) is a typical example. Users need to deal with events and listener models to use these – constructs that rely heavily on a substantial number of advanced language constructs that we do not want to require of beginning students.

Several attempts have been made to simplify graphical user interaction, typically by providing custom GUI libraries with simplified event models. ObjectDraw [2] and Java Power Tools [8] are two typical examples.

These libraries provide a great deal of help, and are steps in the right direction: they take some of the burden of complexity of the programmer, and make it easier for beginners to develop programs that include user interaction.

We believe, however, that we can take this a big step further and remove most of the remaining complexity by integrating the interaction mechanism into our object world framework.

The mechanism we propose is *direct state manipulation*. We will discuss this with an example.

Imagine a Lunar Lander application (a classic small game where the user has to land a space craft by providing the right amount of thrust with limited fuel supplies).

To implement this example, we use two classes: the Lander class that models the space craft, and a Throttle class that models the throttle to control the engine (Figure 5).

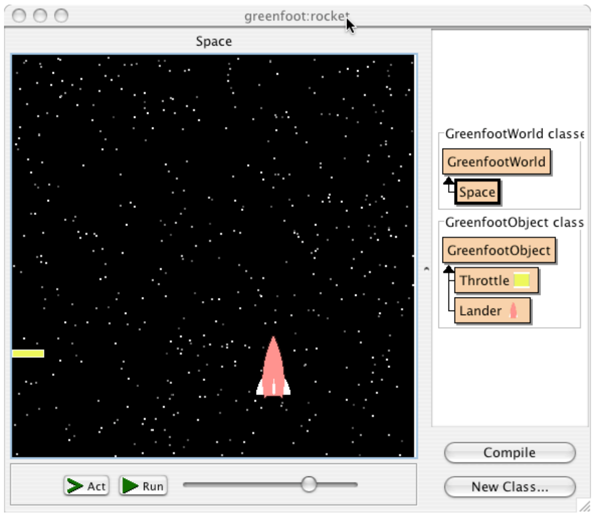

Figure 5: The Lunar Lander scenario

When instantiating a lander, the throttle is automatically instantiated and placed on screen as well (via code in the lander's constructor). The lander will then move according to programmed gravity and the throttle value. This implies that the throttle has, for example, a *getValue()* method to provide its state.

In our greenfoot example, this method simply returns the throttle's vertical coordinate. Users can then, while this application is running, grab the graphical representation of the throttle and drag it up and down on screen. This will update the throttle object's position in real time, thus providing input for the application.

The mechanism is simple to understand and simple to use. The idea is based on directly reversing the animation mechanism: the visualisation framework guarantees that some changes in object state (changes to location, rotation, or icon representation) are immediately made visible on screen. This is what gave us state and behaviour visualisation.

Now we have added a framework that reverses this: user-initiated changes to some of the object's state (here: screen position) are immediately fed back into the internal object state.

This mechanism is inspired by the Squeak framework [6], which provides similar functionality in a very different environment.

When the user drags an object, the framework will request a position change by calling the object's own *setLocation()* method. The object can influence the exact positioning by overriding this method. In the Lunar Lander project, for example, we have modified this method to honour the requested change along the Y-axis, but leave the X-coordinate constant. The result is that the user can freely drag the throttle up and down, but not sideways.

8. I/O WITHOUT I/O

The input mechanism as described here has several advantages. The most fundamental advantage is that the application can receive user input without any specific code written for the purpose of reading input.

The user just writes code that specifies that the rocket should adjust its thrust by the throttle setting, and that the throttle setting corresponds to its Y-coordinate. No additional I/O code is ever written by the user.

This lets students concentrate on the fundamental modelling of object characteristics and interactions, without being distracted by having to write arcane or mysterious event handling code. In an exact mirror of the output model – behaviour can be visualised without programmed output, just by changing the object's location, for example – input can now be received without programmed input code.

Input is achieved just by definition of fields and 'normal' methods, putting the implementation of games within reach of beginning students.

9. CONCLUSION

In this paper, we have mainly discussed use of greenfoot to program simulations or simple games. In reality, greenfoot is not restricted to these categories of applications. It is just as easily imaginable that greenfoot is used to create, say, a virtual drum kit, or an on-screen piano. Any application that profits from graphical output may potentially profit from greenfoot.

Typically, we would envisage that in early examples teachers write some classes for a given scenario (either from scratch or by sharing them with other teachers), and students modify and extend these classes. A little later in a course, students may also create completely new scenarios from scratch. The Karel-The-Robot scenario shown above, for example, consists of only 250 lines of Java code; the Ants example has a total of about 600 lines of code (including empty lines and comments).

Since users program only behaviour in greenfoot, not the graphical I/O code, writing a scenario of this complexity towards the end of a course is not unrealistic.

Overall, we believe that greenfoot may have the potential to allow teachers to use engaging and interesting examples in the classroom, while concentrating on teaching the fundamentals of object-oriented programming: objects, their state and behaviour, and object interaction.

The use of graphical output from the start allows students to get immediate and intuitive feedback about program behaviour. It is also hoped that it helps to create interest and encourage students to experiment and invent modifications and additions to existing programs, especially for students that have a less technical or mathematical background.

The availability of interaction programming allows students to create applications that are closer to the computer games many of them are familiar with. At the same time, the flexible scenarios allow targeting of the application topic to personal preferences, so that the teaching context can be designed to connect to students' interests and backgrounds. We hope that this also increases the level of interest and acceptance in students.

Whether these goals are achieved should be the focus of a study once the first greenfoot versions can be tested in realistic settings.

10. STATUS

An implementation of a greenfoot prototype has been completed and experimentation with this prototype with the goal of functional refinement is currently underway. An early access release of greenfoot is available for free download from *www.greenfoot.org*. A complete system, also to be distributed freely, is expected in the second half of 2005.

11. REFERENCES

[1] Bergin, J., Stehlik, M., Roberts, J., & Pattis, R. Karel J. Robot – A Gentle Introduction to the Art of Object-Oriented Programming in Java. Unpublished manuscript, available [18 March 2004] from: http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html

[2] Bruce, K., Danyluk, A., Murtagh, T., A library to support a graphics based object-first approach to CS 1. In Proceedings of the 32nd SIGCSE symposium, Charlotte, North Carolina, February 2001.

[3] Caspersen, M. E., Christensen, H. B., Here, There and Everywhere – On the Recurring Use of Turtle Graphics in CS1, Proceedings of the Fourth Australasian Conference on Computing Education, 2000.

[4] Dann, W., Dragon, T., Cooper, S., Dietzler, K., Ryan, K., Pausch, R., Objects: Visualization of behavior and state. In Proceedings of the 8th annual ITiCSE conference, Thessaloniki, Greece, 2003.

[5] Henriksen, P. and Kölling, M., greenfoot: Combining Object Visualisation with Interaction, The 19th Annual OOPSLA conference, Educators' Symposium, Vancouver, Canada, 2004.

[6] Ingalls, D., Kaehler, T., Maloney, J., Wallace, S., Kay, A., Back to the future: the story of Squeak, a practical Smalltalk written in itself. In Proceedings of the 12th ACM SIGPLAN OOPSLA conference, Atlanta, Georgia, 1997.

[7] Kölling, M., Quig, B., Patterson, A. & Rosenberg, J., The BlueJ system and its pedagogy. In Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology, 13 (4), December 2003

[8] Proulx, V.K., Raab, j., Rasala R., Objects from the beginning-with GUIs. In Proceedings of the 7th annual ITiCSE conference, Aarhus, Denmark, 2002.

[9] The College Board (Advanced Placement Program), Marine Biology Case Study. Available [September 10, 2003] from: http://www.collegeboard.com/