Objects-First, Algorithms-Early with BotWorld

Jean Griffin
Dept. of Computer and Information Science
University of Pennsylvania
3330 Walnut Street
Philadelphia, Pennsylvania 19104
griffin@seas.upenn.edu

Mark Fickett
Dept. of Computer and Information Science
University of Pennsylvania
3330 Walnut Street
Philadelphia, Pennsylvania 19104
mfickett@seas.upenn.edu

Rita Powell
Dept. of Computer and Information Science
University of Pennsylvania
3330 Walnut Street
Philadelphia, Pennsylvania 19104
rpowell@seas.upenn.edu

Ryan Menezes
rmenezes@seas.upenn.edu

Lu Chen
clu@seas.upenn.edu

ABSTRACT

Inspired by the Pac-Man arcade game and Karel the Robot, we developed a lightweight, graphical "sandbox" called BotWorld for teaching Java with an objects-first, algorithms-early approach. BotWorld has successfully been used by more than one thousand students at the University of Pennsylvania in courses taught by five professors over a three year period. In this paper we describe a series of BotWorld projects and how we use them in our CS1 course, interspersed with non-BotWorld projects. We give examples of algorithms-early projects, projects that encourage creativity, and a project that involves game play and competition. We discuss the process by which we came to develop BotWorld, how we achieve interactivity by using DrJava's "interactions pane", and how BotWorld is "lighter weight" than several notable microworlds. We describe the option to design projects that are "auto-gradable" by software with unit tests which enables hundreds of student programs to be tested and graded in minutes. We provide evidence that the student response as well as the effect on retention have been quite positive. Future work is also discussed. Vistors are welcome to the Botworld web site [1].

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education; D.1.5 [**Programming Techniques**]: Object-oriented programming

General Terms

Documentation, Experimentation, Human Factors, Languages

Keywords

algorithms, BotWorld, competition, CS1, DrJava, game playing, objects first, teaching object-oriented programming, retention

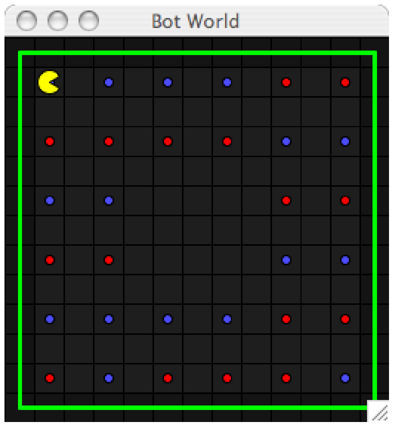

Figure 1: A Bot in a BotWorld, facing east.

1. INTRODUCTION

As CS1 educators and course content developers, we have many demands to balance. We want to make our courses fun enough to attract students, yet rigorous enough so students can succeed if they choose to go on to the next level. We want to engage students with fun technology like graphics and games, but we don't want to bog them down with too much gnarly graphics code. We want our students to be able to "think algorithmically" but we also want them to "think in objects". We want students to realize the sheer power of computing early in the learning process, yet we want them to be able to write code "from scratch". We want to teach the latest and greatest programming languages, but we also want to be sensitive to high school curricula and the logistics involved in a language change. In light of these factors we describe our motivation for developing BotWorld.

2. BACKGROUND

In 2003, due in large part to market demand and declining enrollment, our Computer Science Department changed the language taught in its CS1 course from the functional O'Caml language to Java. The decision was made to use an IDE, but choosing an appropriate one was tricky. Visual Studio and eclipse were excellent, but too confusing for the novices that we hoped to lure gently in to the field. BlueJ and DrJava became our finalists. Both are lightweight, beginner-friendly, multi-platform, and free. Although BlueJ provides very nice graphical representations of classes and objects, DrJava's "Interactions Pane" [2] won us over. The Interactions Pane allows one to "talk directly" to a Java interpreter and get immediate results. It allows one to quickly and easily create objects, call their methods, and observe results without having to create a full-blown application with a main method. This approach is inherently conducive to teaching objects-first. It helps to emphasize the value of building well constructed classes. It supports the concept that a program is a collection of cooperating objects as opposed to a report generator. It is also an excellent tool for giving live demos in the classroom.

Below are some sample interactions that illustrate how the DrJava Interactions Pane works. DrJava provides a prompt (">"). If the user enters an expression or statement, a Java interpreter evaluates it. (If an expression is entered, its value is also displayed.) Note that in the example below it is assumed that a Person class with getName() and setName() methods has already been created and compiled in the Main Pane, which is the area of the screen where the user enters code for classes.

```
> 3 * 5 + 4
19
> Person p = new Person("jo");
> p.getName()
"jo"
> p.setName("mo");
> p.getName()
"mo"
> import java.util.Random;
> Random rand = new Random();
> int x = rand.nextInt();
> x
808944504
```

Six years after adopting DrJava, we still use it extensively in our CS1 course (for both BotWorld and non-BotWorld purposes). We encourage our CS2 students to adopt the eclipse IDE and to install the DrJava plugin because it provides a level of interactivity that is valuable even to professional developers.

Although we don't require a textbook for our CS1 course, we like the "objects-first with DrJava" approach taken by Nino and Hosch [6], especially the use of the Counter class as a first example, and the late introduction of the main method. After using this approach for several semesters we desired to add graphics in a supplementary manner, without making graphics a primary focus of the course, and without abandoning the way manner in which we used the interactive interpreter. By that time we had also adopted the practice of designing many (but not all) of our assignments so they could be "auto-graded" by unit test software. (In courses with over a hundred students, this is truly a blessing!) If we could find find a way to incorporate graphics while retaining the option to auto-grade, that would be ideal.

3. SHOPPING FOR CS1 GRAPHICS

We found several attractive options for incorporating graphics in to a CS1 Java course, but we didn't find one that suited all of our needs. For example, although we found the Karel the Robot system to be quite engaging, it used the traditional "main method mode" of teaching versus the interactive interpreter mode we preferred. Other systems were "too heavy" for us to make a commitment. For example, the event-driven approach described by Bruce et al [3] was appealing but would require a paradigm shift on our part and a substantial commitment to a graphics library and text book, which was daunting. We considered having our students use a graphics library directly (e.g. awt, Swing, Java 2D, Java Task Force library), however we had seen from experience that beginners often get bogged down in the details of graphics code given the opportunity.

In 2005 decided to create our own "virtual sandbox" and call it BotWorld. It would be similar to both Pac-Man (well known and loved by most of our students) and Karel the Robot (a time-honored classic for teaching computer science). Students would interact with BotWorld using the DrJava interactions pane. We would provide games and puzzles that students would complete, and eventually students could create their own. Our CS1 course would not be completely devoted to BotWorld. Rather, professors could pick and choose which topics to explore with it. We would design the system so that even though though there was a graphical component it would be possible to grade some of the projects with software. That would come in handy for our own large courses, and if we ever did an outreach program for high school students, we could accommodate virtually an unlimited number of them. We would design the system so that it would be very easy for beginning students to use right away, but extensible enough to support large, complex, and imaginative projects. We built BotWorld in the summer of 2005 and have been using it successfully ever since. Project and implementation details appear below.

A few notable graphical systems have emerged in recent years. One standout is the approach taken by Guzdial and Ericson [4], which also uses the DrJava interactive interpreter. Although its methodology and supporting materials are excellent at introducing computer science concepts with multimedia, we tried the approach and decided against making such a pronounced commitment to multimedia. We find that although many concepts can be taught within the context of multimedia, in our teaching environment some concepts are more easily explained without it.

Greenfoot is also worthy of mention. Greenfoot is another excellent system that requires a substantial commitment on the instructor's part. Greenfoot provides both microworld environments and an IDE. The microworlds are appealing and the GUI is engaging. A visual class hierarchy is provided, and the user can point and click to instantiate objects and invoke methods. If there is an interactive interpreter mode like DrJava's, where one can type in code versus pointing and clicking, it is not apparent. This feature, which gives students lot of practice with "real coding" is one we find to be very valuable for students who are going to progress in the computer science curriculum. We also prefer having a lightweight IDE that is decoupled from a microworld.

4. BOTWORLD PROJECT SERIES

Currently there are six BotWorld projects, descriptions of which appear below. The introductory project is called BotPlay. The others assume the user has completed BotPlay, but aside from that they are standalone and don't require the completion of any other project.

4.1 Bot Play: Getting Started

In the spirit of game play, our students do the BotPlay project as their very first lab exercise, during the first week of the course. The purpose of this project is simply to play in the sandbox. It effectively engages students to interactively create objects, call methods, analyze return values, and explore javadocs - all for the fun of getting Pac-Man-like *Bots* to move around a *BotWorld*, eat *Dots*, and crash in to *Walls*. Can two Bots occupy the same space? Try it. What happens if a Bot runs in to a Wall? Try it. What can a Bot do besides moving and eating? Look at the javadocs. (We find that students don't just explore the javadocs, they *devour* them, intent on learning how to get the Bots to do as much as possible.) Wait a minute, a Bot can only move one space at a time. That's ridiculous! How can we teach them how to move 5, 10, or n spaces at a time? We'll learn how to teach them how to do that. Students usually very much enjoy this project. The lab room buzzes with activity. Adventurous students examine the javadocs with a fine tooth comb to gain mastery. Sample Interactions:

```
> BotWorld world = new BotWorld();
> Bot bono = new Bot(world);
> bono.move();
> bono.move();
> int x;
> x = bono.getX();
> x
2
> bono.move();
> x = bono.getX();
> x
3
```

In the subsequent lecture, we formally cover the concepts with which the students have already become familiar: objects, constructors, methods, return values, arguments, variables, assignment, etc.. Parallel to the BotWorld approach, we provide various examples of simple classes without graphics, unrelated to BotWorld, and ask students to write similar classes from scratch.

4.2 BetterBot: Inheritance, Modularity, Parameter Passing

Here we provide a gentle introduction to inheritance, with the motivation being to create a better kind of Bot, one that can do new tricks (e.g. turn around or jump). Students can be creative and add tricks/intelligence of their own design. We provide the following code:

```
public class BetterBot extends Bot {

    public BetterBot(BotWorld world) {
        super(world);
    }

    public void turnAround() {
        turnRight();
        turnRight();
    }
}
```

After students try the interactions shown below, we discuss why we can tell a BetterBot to move even though it doesn't appear to have a move() method. Students are encouraged to teach their BetterBot more skills and share their inventions with the class.

```
> BotWorld world = new BotWorld();
> BetterBot sam = new BetterBot(world);
> sam.move();
> sam.move();
> sam.turnAround();
```

4.3 LoopingBot: Algorithms, Iteration, Recursion, Parameters, Modularity

By the time this project is introduced, students are itching to create a Bot that can move multiple spaces at a time, which the provided Bot can not do. This provides good motivation for writing code for a move(n) method with a parameter and a loop (or recursion). Next, students are asked to write a method which commands a Bot to move in an arbitrarily sized series of nested squares, eating all the Dots it encounters. To encourage modularity and code re-use, students are challenged to write a solution consisting of 15 lines of code or less. A nice feature about this project (and MazeBot) is that due to BotWorld's inherent grid structure, students don't have to know about arrays in order to get this early experience with algorithms. This is particularly advantageous because Java's array syntax is cumbersome.

4.4 MazeBot: Algorithms

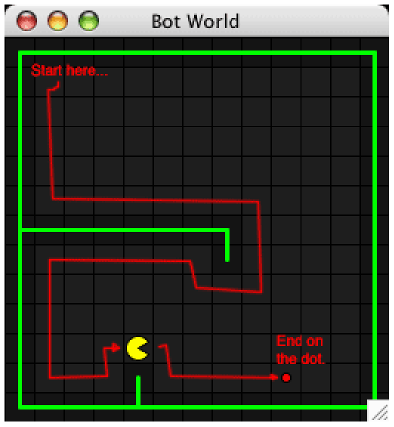

Figure 2: MazeBot

With the MazeBot project, students are asked to encode a classic right-hand wall following algorithm to teach a Bot how to find a Dot. A variety of mazes are provided. As with LoopingBot, knowledge of arrays is not needed.

4.5 FroggerBot: Games, Creativity, Events, Interfaces

In this project students create a version of Frogger, the classic arcade game. They learn about event based programming and use ActionListeners to allow for keyboard control. Specifications are provided for basic features. For example, each student's Frogger class must implement a GameEngine interface and exhibit certain behaviors. Beyond that, students are encouraged to be creative about the look and feel of their game and to add extra features. Many students are motivated by the chance to be creative and respond by encoding interesting visual effects and extra features such as multiple levels.

4.6 TronBot: Games, Algorithms, Competition

TronBot is the newest and one of the most popular projects in the BotWorld series. Although TronBot is another project based on a classic arcade game, its learning outcomes are different from those of the FroggerBot project described above. After building a basic TronBot according to the specifications we provide, the student encodes an algorithm of their own design to give game strategy intelligence to their TronBot. In anticipation of competing against others, students are motivated to encode effective algorithms. The fun begins when the instructional team hosts a live tournament (complete with refreshments and prizes) where the TronBots are pitted against each other, two at a time, until a winner is determined. Similar to the experiences reported by others [5] about computer science competitions, we have found these tournaments to be highly exciting and motivational for everyone involved.

5. IMPLEMENTATION

BotWorld has a Model-View-Controller design. The View is "black boxed" to spare the students from seeing intimidating graphics code, at least in early projects. The Model, as far as the student is initially concerned, consists of two classes: Bot and BotWorld. The first few times we assigned these projects, we gave the students the source code for Bot and asked them to add code to it. However we observed that the students were overwhelmed by seeing a lot of code that they didn't understand, and they managed to break it in all kinds of creative ways! Instead, now we give them a jar and javadocs, and show them how to use inheritance, as illustrated earlier. The Controller can either be a "game engine" object (e.g. FroggerBot, TronBot) or simply the set of commands typed in to the interactive interpreter (e.g. BotPlay, BetterBot).

To set up for the first project, the student installs DrJava [?] and downloads a jar file whose size is approximately 17K bytes. For the sake of simplicity, javadocs for just two classes, Bot and BotWorld, are provided (later on, javadocs that give the students more control are revealed). For subsequent projects, additional jars and javadocs are typically be provided.

6. "AUTO-GRADING"

To facilitate auto-grading, the View could be optional. In other words, the students would see the graphics as visual feedback while testing their code as usual, but the auto-grader would turn off the View and determine correctness by examining the states of the objects as the program executed.

7. RETENTION, STUDENT RESPONSE

8. FUTURE WORK

9. CONCLUSIONS

Ready to share.

Based on our experiences this summer, we have a renewed appreciation for BotWorld as a framework for engaging pedagogy. It provides a fun way to learn object oriented programming.The software is stable, extensible, and serves to develop creative and accurate software developers.

10. ACKNOWLEDGMENTS

The authors would like to thank Fernando Pereira for his help with designing BotWorld, Kevin Lerman for his work on MazeBot, Bertan Aygun and Ari Gilder for their help with the grading software, and Diana Palsetia for her help with the Frogger project.

11. REFERENCES

[1] http://www.seas.upenn.edu/~botworld.

[2] http://www.drjava.org.

[3] Kim Bruce, Andrea Danyluk, and Thomas Murtagh. *Java: An Eventful Approach*. Prentice Hall, New Jersey, 2005.

[4] Mark J. Guzdial and Barbara Erison. *Introduction to Computing and Programming with Java: A Multimedia Approach*. Prentice Hall, New Jersey, 2006.

[5] Iretta B. Kearse and Charles R. Hardnett. Computer science olympiad: Exploring computer science through competition. *ACM SIGCSE Bulletin*, March 2008.

[6] Jaime Nino and Frederick A. Hosch. *Introduction to Programming and Object Oriented Design Using Java*. John Wiley Sons, New York, 2008.