STUDENT COMPETITIONS AND BOTS IN AN INTRODUCTORY PROGRAMMING COURSE[*]

Brian Ladd
Department of Mathematics
St. Lawrence University
Canton, NY 13617
blad@stlawu.edu

Ed Harcourt
Department of Mathematics
St. Lawrence University
Canton, NY 13617
edharcourt@stlawu.edu

ABSTRACT

An introductory programming course that is both an introduction to the major and a university-wide distribution course can suffer from a large disparity of interest as well as ability. Motivating all students to participate improves outcomes. At St. Lawrence University we have found that competitions are fun and engaging for students while providing a vehicle for both individual and collaborative team projects. We use homegrown robot simulation software to draw students into the course with microworld interaction and animation; student competitions based on popular winter sporting events fit our northern clime and improve engagement.

1. INTRODUCTION

St. Lawrence University is a small liberal-arts college with a couple of decades of experience with computer science courses and a seven year old computer science major. Our CS1 course serves as both an introduction for potential computer science majors and as a university-wide distribution course. Due to staffing limitations, it is the only computer course offered that is accessible to non-majors and many students enter with little or no technical background. Our CS1 course is Java-based and students use RoboGames, a Karel-like robot simulator to guide robots (a.k.a "bots") around an arena to perform some task (e.g., searching for items, maneuvering around walls, etc.). RoboGames takes its name from the repertoire of competitive assignments we have designed to motivate student engagement with the course material.

---

[*] Copyright © 2005 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

The RoboGames are loosely modeled on winter sports, an important part of student life at St. Lawrence University. Intra-sectional and inter-sectional competitions include aesthetically graded figure skating, a timed biathlon, and a head-to-head hockey competition. From the beginning of the course, the games are mentioned on a regular basis; as each new feature of the microworld is revealed, its relationship to the various events is spelled out.

During the recent change to Java and an object-oriented CS1, the decision was made to modernize the topics as well as the tools. Historically, student programming projects were procedural, building toward individually designed video games as the culmination of the course. While successful in its time, the video game project was harder to do in Java and it was difficult to design assignments that imparted the necessary skills (drawing, animation, input processing) at an appropriate level for CS1 students. Starting with a clean slate, we devised RoboGames.

Our primary goals in redesigning the course were to provide a strong basis for students going on to CS2, to make the material engaging, and to make the course attractive to non-majors. RoboGames provides an entertaining introduction to programming topics such as selection and iteration. The competitive spirit pushes students to develop exactly the skills they need at the end of a CS1 course.

Attracting non-majors to CS1 is a challenge; programming is hard work. Additionally, there is some evidence suggesting that one of the reasons computer science is less attractive to females is that computer games involve more "male-oriented sports and hobbies" including games that involve "wars, battles, crimes, and destruction" [9]. Our competitions are non-violent (bots never shoot or damage one another) and are based on sports that enjoy strong participation across gender lines. All together, RoboGames seems to meet the course design goals.

The rest of the paper examines the motivation for our games, how we use them, and their strengths and weaknesses. Section 2 discusses related work. Sections 3 and 4 present an overview of our development environment and the competitions. Section 5 draws some conclusions and provides an evaluation of what we have found in our past five semesters about our use of bots and competitions.

2. RELATED WORK

Diversity in approach is a hallmark of the introductory computer science curriculum [12]. The only real agreement seems to be that motivating beginning students is a path to more active learning [16, 8]. One motivational tool that has been used with some success is the idea of programming simulated (or real) robots. This section surveys uses of animated robots closest to our own and then looks at other reports on the use of competition in computer science.

2.1 Robot Simulation for Learning

Robots have stirred the imagination since before their name was coined. In computer science education, first there was Karel [15]. Karel provided an environment in which the user-programmed bot was able to detect walls to avoid and beepers to manipulate (pickup or put down). This is the inspiration for our SimpleBot environment described below. The use of a limited number of functions (e.g. turnLeft without a corresponding turnRight) provides early motivation for the inclusion of user-defined functions.

Karel has been revisited in each succeeding language and curriculum revamp in the intervening decade. Java-based robots have been used in courses running the gamut from traditional, procedural approaches [1] to those with an objectsfirst [18, 19] and even some exploring an inside-out strategy for exposing cognitive complexity only as students advance [2]. These approaches vary in how complex the virtual environment is. Most are either direct inheritors of Karel's simple, intersection-based microworld or are based on bot movement in a completely free-form way. Our work differs from them in that it provides both interfaces for the programmer (with additional complexity being exposed as the student progresses).

Simulated bots are also found outside of academia. MindRover: The Europa Project [5] is an entertaining game based on building and visually programming robots to perform different types of tasks (destructive competition, tag, racing). The competitive nature of the sports-based tasks informed the development of our games. The visual interface of MindRover makes it an accessible game but lacks an obvious path to the use of traditional programming languages. The massive paradigm shift was judged to be too jarring to consider MindRover for use in a CS1 course.

IBM's AlphaWorks developed its own Java-based robot environment: RoboCode [10, 11]. Programmable tanks compete to annihilate one another in single or team matches in large, open arenas. While RoboCode inspired our architecture and spurred the actual development of RoboGames, the results are quite different. Our worlds are more complex, containing many non-bot obstacles; our competitions are less "violent", modeled on winter Olympic events rather than combat.

Other microworld environments have been used for teaching introductory computer science [17]. The use of robots, real and simulated, has been found to be beneficial to student learning both anecdotally [7] and more formally [6].

2.2 Competition in CS1

Previous experience crafting video game programming projects led us to fear that bot-to-bot violence would alienate some students including many women. Some published work concurs [3, 4, 13]. Some of the gender-specific studies also claim the competitive nature of video games in general alienates women. We are happy to report in this paper that this has not been our experience.

A survey of the literature found some support for our suspicion that competition could be harnessed to motivate introductory programming students. [16] offers competition as one strategy for improving individual outcomes in CS1 and [14] describes success in shaping an introductory course around a large competition at the end. [14] discusses using grades as a reward for success in intra and intersection competition; we discuss our considerations on this point when evaluating our success using competition.

3. THE ROBOGAMES ENVIRONMENT

RoboGames is our homegrown robot simulation environment. The current implementation of the program provides a top-down view of the action. A typical assignment consists of an instructor-provided "world" and a definition of a "problem" that must be solved. Students design and implement a robot class to solve the problem by extending the Bot (or SimpleBot; see below) class. Running the world then instantiates their bot, starting execution with their run method. This section provides an overview of the run-time and class architecture of RoboGames; our use of both simple and free-range bots is justified at the end.

3.1 RoboGames Architecture

RoboGames is a discrete-time 2D physics simulator. Bot-based components in the world have plug-in architecture that permits programmers to specify behavior. When a simulation begins, the Games instantiates the model world (a Stadium) and connects a StadiumView-based renderer to the world. While we do not yet take advantage of it, this architecture permits us to change the renderer (or view) of the model world without having to modify the simulation itself.

Mobile content objects (e.g. Bots and Balls) each run in their own Java thread. On each tick of the clock, the Games awakens each thread; each thread runs for one simulation defined time quantum or until it calls a blocking function. Methods that interact with the simulation are blocking: moving, using sensors, picking up or dropping objects. The basic architecture is shown in Figure 1.

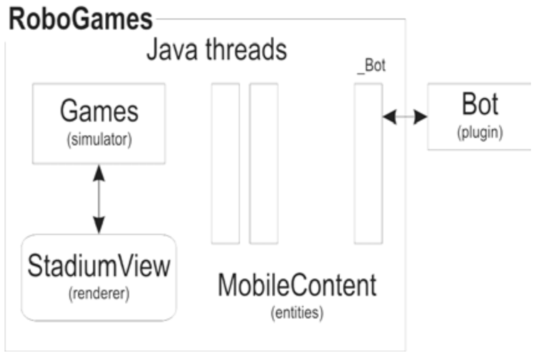

Figure 1. RoboGames Architecture

This complex architecture is almost completely hidden from students. After installing the appropriate .jar file (already done for them in the computer labs), a student constructs a bot by extending the Bot class. When a world is executed, the run method of the student's Bot is the behavior of a proxy object running inside the simulation. Interaction with the environment is routed through the proxy while behavior comes from the student's object. This architecture permits us to protect the simulation from modification (intentional or otherwise) by student code; competitions run fairly.

In addition to mobile content, the Games simulates walls, goals, and beepers. A wall simply blocks a bot's progress (and line of sight). Goals are similar to walls except that they interact with balls to record a score (across the correct edge of the rectangle). Beepers, represented as colored disks on the screen, serve as different objects in different assignments: targets to shoot, keys to find, flowers to plant, or pizzas to deliver.

3.2 Two Versions: Bot and SimpleBot

We have two versions of our bots, simple bots and free- range bots. Students begin the course by implementing simple bots (by inheriting from the class SimpleBot which, as shown in Figure 2 extends the free-range Bot). SimpleBot movement is limited to intersections on a grid; simple walls and beepers are similarly limited to the grid. SimpleBot is the spiritual successor of Karel. Using SimpleBot permits students to learn to use objects and to see the syntax for extending objects; students are introduced to traditional programming concepts such as iteration, selection, and the use of methods.

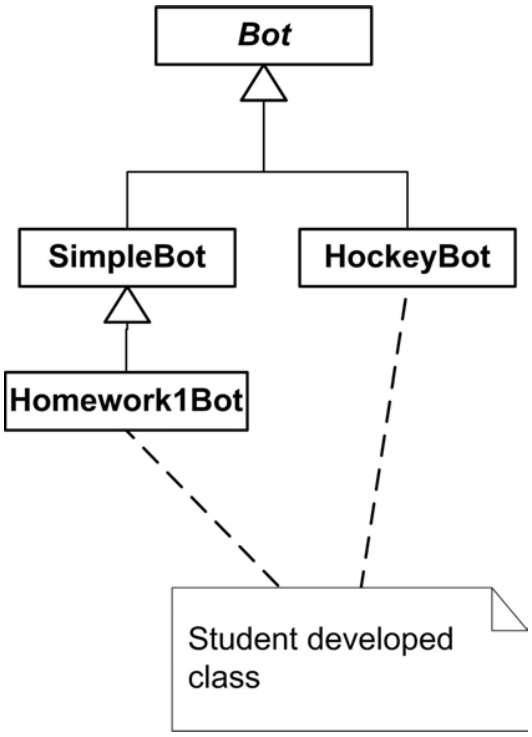

Figure 2: Bot class hierarchy.

Later in the course, students extend the Bot class directly to implement free-range bots. Free-range bots can turn to arbitrary facings, move arbitrary distances, and observe their environment at a distance. The Bot class's added complexity comes at the point where most students are noticing the constraints put on them by the SimpleBot. All RoboGames events are run using free-range bots.

We include both a simple and free-range version of bots for the same reasons inside-out programming [18, 19] hide and then expose the complexity of their bots: students can focus on the task of learning Java syntax and basic loop and selection structures without facing concerns about angle and distance calculations. Then, when they have begun to internalize the basics, the added features of the free-range bot stimulate them with new possibilities.

4. STUDENT COMPETITIONS

Our introductory curriculum is somewhere between traditional procedural programming and objects-first. Students spend the first half of the semester learning basic programming skills by solving increasingly difficult problems with bots. From day one, however, they are regaled with tales of the upcoming RoboGames. When switching from Karel-like to free-range bots, many lab assignments are designed to give students experience with worlds like those they will see in competition (e.g. finding a puck, chasing it, and putting it into a goal).

RoboGames events are divided into easy and hard events. Each student must participate in at least two events and at least one hard event. Programming teams may only enter hard events.

Two-thirds of the way through the semester students turn in their contenders. The bots competing in the various games (see below) face off in-class for the right to represent the section in the inter-sectional, "international" games. The finals are held during the evening in a large lecture hall on a large projection screen. Scoring and programming challenge are specific to each event. The fact that we don't require attendance, yet typically have 90% turnout (including students without their own bot in contention) suggests that students find RoboGames interesting and worthwhile.

Comments from other departmental faculty indicate that students are quite engaged in these games. As with early programming assignments the events in RoboGames were designed to emphasize different programming concepts. An effort was made to vary the challenge and the appeal of events for different types of students. Some events are also designed with team programming in mind. The remainder of this section describes typical RoboGames events, discussing the design goals and scoring particulars of each event.

4.1 Figure Skating

Free-range bots retain the ability to drop and pickup various colors of beeper and to turn on and off a "looking-beam", a ray that marks where the bot is looking when it applies its sensors. These features, combined with artistic pathing and lots of spins, are used in the figure skating competition to permit student competitors to use their aesthetic sense.

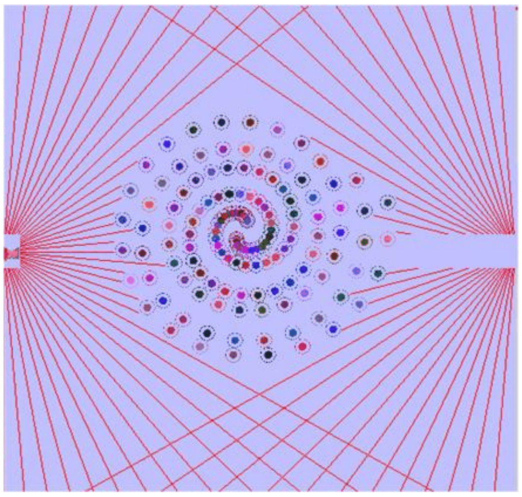
Figure 3: A figure skating champion.

Figure skating bots complete programs between three and five minutes. These limits ensure that students write enough to justify their grade and that the competitions do not last too long. Figure 3 shows the ending configuration for one of the past winners of the figure skating competition.

Figure skating scoring is subjective. A panel of judges, usually other departmental faculty, rate the beauty of the source code and the performance on a scale of one to ten; the highest combined score receives the gold.

Figure skating is the easiest competition. It emphasizes basic software engineering concepts (e.g. good code formatting, commenting, and modularity) for the source code evaluation portion of the score as well as basic programming concepts (e.g. selection, iteration, delegation) for the performance evaluation. Figure skating also lends itself to modifications that emphasize other concepts: requiring bots to be driven by data files adds file processing; requiring the bots to perform the same skating forward and backward adds the use of arrays.

Students use all available creative freedom in the face of subjective scoring. Artistic students create interesting, colorful patterns while mathematically inclined students plot interesting functions using beepers. One of this years finalists created a Red Sox logo in such a way that the final design wasn't evident until well into the routine.

4.2 Biathlon

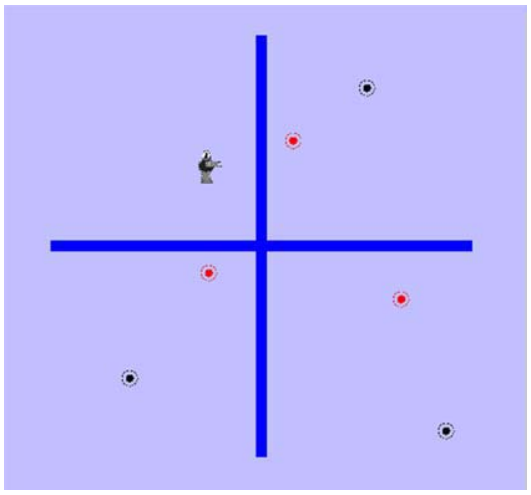
Figure 4: Biathlon bot.

In the biathlon, a bot must "ski" to three different locations, find a shooting stand (a black beeper) and then find and "shoot" (using a looking beam) a target (a red beeper). The bot must complete a circuit back to its original location. Scoring is based on the best time recorded on two runs. The exact locations of the stands and targets vary from run to run so rapid search and shortest path calculations are emphasized in the event. Figure 4 shows the initial setup of a biathlon bot.

Biathlon is another "easy" competition (only individual programmers can enter). The repetition of the same task lends itself to good use of methods; the rotation of relationships between the shooting stand and target leads competitors to use parameterized methods. Mathematically inclined students use trigonometry to compute angles and to find and move the shortest distance between two points.

Searching for a beeper is an algorithmic problem involving a technique similar to a bisection method (see Figure 5) where a bot does a quick scan every Δ degrees. If the disk is not found at that Δ then the search needs to be done at a smaller Δ, usually Δ /2. Biathlon can be modified to emphasize arrays by requiring the bot to place three beepers in the original quadrant in the same relative positions where the targets were in their quadrants.

```
class Homework1Bot extends Bot {
    public void findBeeper() {
        double spin = 0;  // total spin with current search angle
        double delta = 5; // amount to turn between looks
                          // use sensor to find beeper
        while (!look().getWhat().equals("Beeper")) {
             turn(delta); // no beeper = turn to keep looking
             spin += delta;  // track turn
             if (spin >= 360) {      // turned full fruitlessly
                   delta /= 2;       // refine search interval
                   spin = 0;         // track turn from 0 again
                   }
             }
        } // findBeeper
    public void run() {         // every bot has a run method
        findBeeper();
    }
}   // class BeeperBot
```

Figure 5: Beeper search method.

4.3 Hockey

Hockey makes use of the multi-bot competition capabilities of our simulator. Two bots enter a world with two goals (one for each to defend) and a puck. Games are played to the first score and students play a single-elimination tournament of best of three (or five in the finals) competitions. Figure 6 shows an initial setup of a hockey game.

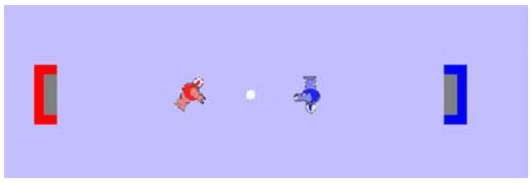
Figure 6: Hockey

Hockey is a "hard" competition. The puck is a moving target that must be driven into a specific face of the goal to score; the other bot is a moving obstacle that must be avoided in order to score. This event was designed to require a great deal of effort to devise a winning strategy. Students spend the most time plotting and refining their strategies. Winning teams often utilize features of the simulation that are beyond those covered in the classroom. Exploration of the environment and documentation is encouraged from the beginning of the course; it is richly rewarded in the RoboGames.

These typical events show the breadth of challenges available when we develop competitions. We are always designing new events (e.g., we have used speed skating and tag events in the past and we are working on a variation of the Pong video game for the next games) to keep the competition fresh, and to prevent a single winning strategy from taking root in the student culture thereby removing balance from the games.

5. EVALUATION

Bots and competitions have enabled us to create a studio based teaching environment where students are engaged and motivated. Students collaborate more effectively when they share a common problem domain and software infrastructure. Not without its problems, bots and competitions create extra work for instructors as well as presenting other pedagogical and support issues.

Bots provide a common problem domain and vocabulary. Our CS1 students come from diverse academic disciplines. Our bots and competitive events provide a common background where all students start on a level playing field. Bots allow us to concentrate on core programming concepts unfettered by some of the material presented early in many Java textbooks (e.g., applets, graphics, HTML, I/O, the JVM, etc). One downside of this is that we have yet to find a textbook that complements our CS1 syllabus of concentrating on core programming concepts. In practice lack of textbook support has not proven to be a burden. Course notes along with an ample supply of sample programs suffice in providing students with enough material to work independently.

One caveat of using a microworld such as our bot environment is the potential to isolate students from programming in the real world. We are careful not to teach bots for the entire course. We tend to use bots for explicating and exercising core programming concepts. Conventional stand alone programs are taught the last third of the semester.

There is ample evidence that active-learning environments promote better understanding and retention of concepts as well as increasing student retention within computer science programs. We have found that bots support an active learning pedagogy. Class periods are no longer purely lecture or lecture/lab but a mix of problem solving, programming, lecture, and lab.

Competitions with bots motivate and engage students. We have seen a level of student excitement and anticipation using bots and competitions that we have rarely seen using more traditional programming assignments. We often see students discussing strategies and tackling problems trying to devise a winning solution. Students watch with anticipation what a bot is going to do (or not do), yell across the room, cheer, and applaud. One downside of competitions is that when there are winners there are also losers; disappointment is sometimes present. Our events are decidedly open-ended allowing motivated and more able students to put as much work in their bot as they desire.

Competitions support collaborative work and team projects earlier in the computer science curriculum. While this is not necessarily unique to bots it has been easier for us to get students more involved with collaborative work and team projects.

Bots allow us enough flexibility in our pedagogy to teach CS1 with either a procedural or objects-first approach. While we have yet to fully embrace an "objects first" approach our pedagogy splits the difference between purely procedural and object oriented programming. We introduce objects early by having students create instances of predefined classes, (e.g., bots, balls, walls, beepers). While students implement at least one class other than the class that contains main student designed classes are few. Once a student begins developing their bot's behavior in the run method of the bot the programming paradigm is largely procedural.

In order to have a successful inter-class competition it is imperative that the lecturers teaching the sections communicate and synchronize more often than they would normally need to.

Our measure of success and goal in introducing bots and competitions in CS1 has been to increase the level of engagement and motivation in our students as well as provide a course suitable and attractive for majors and non-majors alike. In this regard we have been successful. Competitions both challenge and motivate students. We have used bots to create a more studio-based class environment which in turn has made students more engaged and attentive. Bots provide a level playing field for all students while providing an excellent vehicle for running RoboGames. While there was a substantial startup cost in creating labs, lectures, and assignments customized to our bot microworld these are in a large part reused and refined, ameliorating the cost over time.

6. SUMMARY

As we look to next semester and beyond, we consider how to leverage our use of RoboGames. It is a generic game-building architecture (for a simple, 2-dimensional game); it makes a good jumping off point for students in our new game design course. There are also a large number of technical improvements that we want to make, many of which are suitable for student senior projects: replace the renderer with a real 3D engine; improve the debugging interface; rewrite the physics engine to improve collision detection; add a third dimension to the arena. We realize that changing the courseware is only a small part of changing the course.

This paper reports our satisfaction with the pedagogical success of RoboGames. We are working to devise a methodology to measure our success. Quantified outcomes will permit us to tweak the course and compare different structures and uses of the package.

Designing an introductory programming course for students with diverse backgrounds is difficult. Motivating students is necessary to get them to engage with the material; robot simulators and, in particular, an Olympic-style bot competition have successfully motivated our CS1 students.

7. REFERENCES

[1] B. W. Becker. Teaching cs1 with karel the robot in java. In ACM SIGCSE Bulletin, Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education. SIGCSE, ACM, February 2001.

[2] D. Buck and D. J. Stucki. Jkarelrobot: a case study in supporting levels of cognitive development in the computer science curriculum. In ACM SIGCSE Bulletin, Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, February 2001.

[3] M. Chaika. Ethical considerations in gender-oriented entertainment technology. Crossroads, 2(2):11—13, November 1995. http://www.acm.org/crossroads/xrds2-2/gender.html.

[4] M. Chaika. Computer game marketing bias. Crossroads, 3(2):9—12, November 1996. http://www.acm.org/crossroads/xrds3-2/girlgame.html.

[5] CogniToy, LLC, Acton, MA, USA. MindRover: The Europa Project, 1.07 edition, 2000. http://www.mindrover.com/mindrover/mindrover.htm.

[6] B. S. Fagin and L. Merkle. Quantitative analysis of the effects of robots on introductory computer science education. Journal on Educational Resources in Computing (JERIC), 2(4), December 2002.

[7] M. Goldweber, C. Congdon, B. Fagin, D. Hwang, and F. Klassner. The use of robots in the undergraduate curriculum: experience reports. In ACM SIGCSE Bulletin, Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, February 2001.

[8] C. Hill. A media enhanced introductory programming class. Journal of Computing Sciences in Colleges, 18(1):134—143, October 2002.

[9] S. Kiesler, L. Sproull, and J. S. Eccles. Pool halls, chips, and war games: women in the culture of computing. ACM SIGCSE Bulletin, 34(2), June 2002.

[10] S. Li. Rock 'em, sock 'em robocode: Round 1. IBM Developer Works, January 2002. http://www-106.ibm.com/developerworks/library/j-robocode/.

[11] S. Li. Rock 'em, sock 'em robocode: Round 2. IBM Developer Works, May 2002. http://www-106.ibm.com/developerworks/library/j-robocode2/.

[12] T. Long, B. Weide, P. Bucci, D. Gibson, J. Hollingsworth, M. Sitaraman, and S. Edwards. Providing intellectual focus to cs1/cs2. In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education, pages 252—256. SIGCSE, ACM Press, 1998.

[13] M. Natale. The effect of a male-oriented computer gaming culture on careers in the computer industry. ACM SIGCAS Computers and Society, 32(2):24—31, June 2002.

[14] R. P. Pargas, J. C. Lundy, and J. N. Underwood. Tournament play in cs1. In ACM SIGCSE Bulletin, Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education, March 1997.

[15] R. Pattis. Karel The Robot: A Gentle Introduction to the Art of Programming. Wiley, 1994.

[16] E. Roberts. Strategies for encouraging individual achievement in introductory computer science courses. In ACM SIGCSE Bulletin, Proceedings of the thirty-first SIGCSE technical symposium on Computer science education, March 2000.

[17] S. Rodger. Introducing computer science through animation and virtual worlds. In Proceedings of the 33rd SIGCSE technical symposium on Computer science education, pages 186—190, Cincinnati, KY, USA, 2002. SIGCSE, ACM Press.

[18] D. Sanders and B. Dorn. Classroom experience with jeroo. Journal of Computing Sciences in Colleges, 18(4):308—316, April 2003.

[19] D. Sanders and B. Dorn. Jeroo: a tool for introducing object-oriented programming. In Proceedings of the 34th SIGCSE technical symposium on Computer science education, pages 201—204, Reno, NV, USA, 2003. SIGCSE, ACM Press.