Using artificial intelligence to help bridge students from high school to college

Elizabeth Sklar[1,2], Simon Parsons[1,2], Sheila Tejada[2], Susan Lowes[3], M Q Azhar[1], Samir Chopra[2], Richard Jansen[2], and Ira Rudowsky[2]
[1]Dept of Computer Science, Graduate Center, City University of New York, New York, NY 10016 USA
[2]Dept of Computer and Information Science, Brooklyn College, City University of New York, Brooklyn, NY 11210 USA
[3]Institute for Learning Technologies, Teachers College, Columbia University, New York, NY 10027 USA
contact author: sklar@sci.brooklyn.cuny.edu

Abstract

This paper describes work from the **Bridges to Computing** project at Brooklyn College of the City University of New York. This project focuses on the transition from high school to college with the intention of encouraging more students to study some aspect of computer science. The Bridges project has both introduced new undergraduate courses into our computer science curriculum and revised existing courses, as well as developed activities for high school students to help better prepare them for college-level computer science. Here, we report on the use of ideas from artificial intelligence implemented within several of these interventions.

Introduction

The **Bridges to Computing** project at Brooklyn College of the City University of New York (CUNY) focuses on the transition years from high school to college, working to better inform students about and prepare them for careers in computing fields. **Bridges** involves components geared toward advanced high school students, early college students and advanced college students. Project activities include:

- **formal training**—via context-based introductory and interdisciplinary undergraduate courses;
- **informal training**—through after-school and summer programs for high school students;
- **mentoring**—from high school students to undergraduates to graduate students and faculty; and
- **community outreach**—to the College community and beyond, by connecting undergraduate computing students with local schools, small businesses and campus academic departments.

We describe the activities that have occurred during the first 18 months of the project, focusing on the academic components where the use of *artificial intelligence (AI)* is emphasized as a means of engaging students. Thus far, the Bridges project has reached approximately 80 high school students from Brooklyn public schools, and at Brooklyn College: 12 advanced undergraduate computing student *Ambassadors* (i.e., peer mentors), and over 500 undergraduate students through 17 sections of 12 newly developed or updated computing courses.

This paper is organized as follows. First, we describe the project's academic programs, describing the formal, classroom-based activities for undergraduate students and the informal, after-school activities for high school students. Then we detail the curricula which include topics within the field of artificial intelligence. Next, we present evaluation results from each of the academic activities. Finally, we conclude with a summary and directions for future work.

Academic Activities

The project's academic activities are centered around the philosophy that a broader cross section of students will be attracted to computer science (CS) if it can be demonstrated to be relevant to students' daily lives and future careers, helpful to students' communities and/or interdisciplinary in one or more ways (Moskal, Laisch, & Middleton 2001; Sleeter & Grant 1987). Accordingly, the formal and informal training components of the Bridges project are structured around five "flavors", emphasizing the intersection between computer science and: (1) business, (2) law, (3) medicine, (4) graphics, and (5) robotics. As discussed below, the last three flavors in particular have produced curricula that take advantage of AI-based solutions. The formal training components have involved the creation of new and updated undergraduate courses in two categories: introductory and interdisciplinary computing. The informal training components have involved a summer program and an after-school computing preparatory class, both for high school students.

Formal training: Introductory Computing

The undergraduate introductory computing curriculum at Brooklyn College consists of three courses, which can generically be referred to as *CS0*, *CS1* and *CS2*. For each of these courses, multiple sections are offered, adhering to the College's philosophy of keeping class sizes small. Here, we refer to "Bridges" and "Non-Bridges" sections of each course. The *Non-Bridges* sections are those that follow the department's traditional syllabus, typically presenting examples that are either abstract (i.e., without an applied context), mathematical (e.g., write a program to compute the Fibonacci sequence), or classic textbook tasks (e.g., write a program to compute change or implement an address book). The *Bridges* sections, while necessarily covering the same

concepts on the standard syllabus, select examples from within a single themed context—one of the flavors listed above. Within each of these flavors, particular emphases have evolved, particularly in the CS0 classes: e-commerce within the business flavor, cryptography within the legal flavor, biologically inspired systems within the medical flavor, and games within the graphics flavor.

The *CS0* course is part of the College's "Core Curriculum" requirements in computing and mathematics. The course is designed to give students with no computing background an introductory-level exposure to a cross section of topics within computer science and provide them with some hands-on experience with computers and programming. The topic areas include: basic architecture of computers and networks, algorithms and computer languages, data representation and storage, event-driven programming, control structures (branching and looping), solvability and feasibility, and security, privacy and encryption. The *Non-Bridges* sections of CS0 use HTML and Javascript to teach basic programming concepts (such as control structures). The *Bridges* sections employ special interest programming environments to introduce basic programming concepts — RoboLab (Erwin, Cyr, & Rogers 2000) for robotics, NetLogo (Wilensky 2002) for medical applications and Scratch (Peppler & Kafai 2007) or Alice (Cooper, Dann, & Pausch 2003) for graphics and games.

Many sections of CS0 are offered each semester (over 20), totalling between 400-500 students. These courses are typically taken by freshmen or sophomores, students who have not declared a major or students who have decided not to major in computer science[1]. The drop-out rate in CS0 is typically not high, since students need to complete the course at some time during their college career. What is more unusual is for students to decide to major in computing as a result of taking this course and then go on to take CS1 in a subsequent semester. One of the goals of the Bridges project is to increase the number of students who take CS1 after successfully completing CS0.

The *CS1* course is the first programming course taken by students who intend to major in computer science. It is also required by a handful of other science majors. Most students attending Brooklyn College graduated from public high schools in Brooklyn and had limited access to computer science courses; some took "technology" courses that introduced students to a few Microsoft Office applications (e.g., Excel), but the overwhelming majority did not have the option of taking a programming course in high school. CS1 is taught in C++, and the topics covered include: data and output, control structures and input, functions, arrays and strings, sorting and searching, and simple classes. CS1 is taught using either a Microsoft Windows or Mac OS X environment, employing an integrated development environment such as Dev C++[2] or CodeBlocks[3]. Many students have little experience with computers and are not familiar with file systems, the DOS command window, network concepts, etc., so it is also necessary to include much of the basic information conveyed in the CS0 course.

CS1 tends to have a larger drop-out rate, as compared to CS0. The published national average indicates that only 50% of students opt to continue in computer science after taking their first course in the major. At Brooklyn College, during the period Fall 2000 through Fall 2006, the retention rate in CS1 is 39%. Figure 1 shows the percentage of students who took CS2 after taking CS1 (labeled "CS1 retention") and the same for students who stayed with the major after completing CS2 (labeled "CS2 retention"). One of the key issues we have identified through surveys administered as part of the Bridges program is that students are ill-informed about the differences between CS0 and CS1, so many students enroll in CS1 when it would have been more appropriate for them to take CS0. Another goal of the Bridges project is to improve retention of students in CS1, not only decreasing the drop-out rate within the CS1 term but also increasing the number of students who subsequently complete CS2.

The *CS2* course is the second programming course taken by students who intend to major in computer science. It is also taught in C++, and it introduces UNIX, using either a Linux or Mac OS X environment, including a smattering of basic commands (like `ls`, `cd`, etc.). Students are shown how to use a text editor, such as emacs, vi or pico, and how to compile (using g++, the Gnu C++ compiler[4]) and run from the UNIX command-line. They are introduced to file system concepts; and the relationships between the familiar graphical desktop and the underlying file structure are explained. Topic areas include: classes and object-oriented programming concepts, specifications and testing, pointers and arrays, recursion and templates. The CS2 course also has a significant drop-out rate, and, as above, one of the goals of the Bridges project is to improve retention of students through CS2 into the rest of the computer science major.

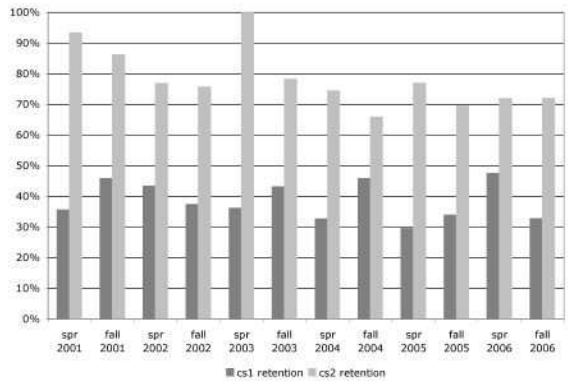


Figure 1: Retention rates at Brooklyn College for CS1 and CS2, from Fall 2000 through Fall 2006

[1]Students who know they want to major in computer science take CS1, which also fulfills their core curriculum requirement in computing and mathematics.

[2]`http://www.bloodshed.net/`

[3]`http://www.codeblocks.org/`

[4]`http://gcc.gnu.org/`

Figure 2: Students from the high school programs

Formal training: Interdisciplinary Computing

As part of the Brooklyn College core curriculum, advanced students who have already chosen their major are required to take two interdisciplinary courses. We have developed and implemented an interdisciplinary course called *Exploring Robotics* as part of this effort and connected to the Bridges Project. This course was offered for the first time in Fall 2006 and has proven to be tremendously popular. At an average of 20 students per section, we have offered 5 sections in Fall 2006, 1 in Winter intersession 2007, 5 in Spring 2007, 3 in Summer 2007 and 8 in Fall 2007.

Informal training: Summer Institute

In July 2006 and July 2007, we ran 8-day free summer programs for high school students. We recruited students from local public high schools in Brooklyn and approximately 35 students attended each summer. The goal was to give students who have limited or no access to computer science courses in their high schools an opportunity to learn about the field, its broad applications and interdisciplinary nature, and to gain hands-on experience with 1-2 technologies. We divided the eight-day institutes into 3 "taster" days and 5 "pick" days. During the *taster* days, students attended 5 half-day sessions, one for each of the five Bridges flavors. In the afternoon of the 3rd day, they selected one flavor to concentrate on for the remaining *pick* days; the rest of that afternoon was spent on a community-building activity, away from computers but using a computer-based theme. The pick days culminated in an evening *Showcase* where students invited family and friends to see posters and demonstrations of what they had built. More than half of the students brought guests. Feedback from parents was tremendous. Parents were pleased that their children had been able to come to a college campus and interact with faculty in a friendly, non-threatening environment. They hoped the experience would help their children be more motivated to apply to college.

Informal training: Computing Preparatory Course

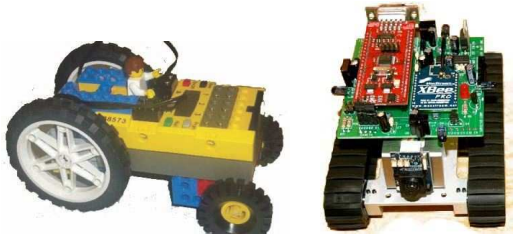
In Fall 2006 and Fall 2007, high school students were invited to attend a Computing Preparatory Course during after-school hours. Some of the high schools have given students high school credit for attending the class[5]. The intended purpose of the class is to give students more in-depth experience with the topics introduced during the summer; in practice, there are many who attend the Computing Prep course who did not attend the Summer Institute. The sessions are lab-based, so students can work at their own pace. Undergraduate mentors assist, which facilitates the largely individually-tutored environment. The course is structured so that approximately every 4-6 weeks a new topic is introduced, again following the five Bridges flavors. Topics covered include: HTML and Javascript, cryptography, simulations using NetLogo, robotics using RoboLab, games using Scratch or Alice.

AI-centric Curricula

This section details the three flavors that have broadly introduced *artificial intelligence*: robotics, biologically-inspired simulations and multi-agent games.

Robotics and Agents

At all levels, undergraduate and high school, students are introduced to the notion of artificial intelligence through the *intelligent agent* paradigm. An agent is an *automonous* entity that exists in some kind of *environment*, either virtual or physical. It receives inputs through sensors that perceive properties of their environment and/or themselves, and it generates output through actuators that effect change on their environment and/or themselves. The AI is the part that comes in between receiving input and generating output—this is where something *intelligent* should happen. Students are intrigued by the notion that they can construct sets of rules that govern the behavior of an agent. They are introduced to concepts such as *state* and Markov processes, rule-based systems and probabilistic reasoning.



(a) LEGO Mindstorms RCX (b) Surveyor SRV-1

Figure 3: Robot platforms

In the high school components and the CS0 course, LEGO Mindstorms robots are employed (see figure 3a). Students are taught about simple sensor inputs (e.g., light level and bump), how physical properties can be translated into numeric values and how those numeric values can be input to a program that emulates intelligent behavior on the part of their agent. They are given a variety of tasks designed to introduce them to: the RoboLab[6] programming environ-

[5]The same arrangement is being negotiated for all participating high schools.

[6]`http://www.ceeo.tufts.edu/robolabatceeo/`

ment, the design-write-test-debug software develop cycle, basic programming concepts such as branching, looping and data storage, and basic computer and robot hardware concepts such as memory, power, sensors and motors.

In the CS1 and CS2 courses, students' exposure to robotics is primarily through examples and simulated robots (virtual agents), though both classes are given at least one assignment using a physical robot. The Surveyor SRV-1[7] is currently being used. This small, reasonably-priced robot has an on-board web camera and is controlled from a laptop via radio communication (see figure 3b). The classes each meet for 4 hours per week, with lecture and lab sessions alternating. An example of a task for a simulated robot is one in which students must devise a control algorithm for a robot that can move around in a virtual 2-dimensional grid, using commands such as "left", "right", "up" and "down". The robot has a fixed amount of fuel and expends some of its energy with every command. The robot's world is inhabited with randomly placed pieces of "treasure", and students' controllers should maximize the amount of treasure captured by the robot before it runs out of energy. This task is assigned in both CS1 and CS2 courses, but the programming requirements are different. For example, in CS1, students use a 2-dimensional array of characters to store the robot's world; whereas in CS2, students must create several classes to represent the robot and its world. Students are exposed to basic AI concepts, such as state, decision trees and search strategies.

Biologically-inspired Simulations

The overarching idea behind the Biomedical computing flavored sections of CS0, CS1 and CS2 is to show how computers can be used in fields that are broadly related to medicine. In the flavored sections of CS1 and CS2, for example, students work on several examples that manipulate patient records and DNA sequences rather than the more traditional examples. Especially in CS1, which is a required course for many science majors, the idea is to get students to appreciate how computers, and in particular programs that they can write, might be used in the pursuit of the science they are studying. Across all three courses, however, the bulk of the examples that the students work on are agent-based simulations of small biological worlds. We deal with simple agent models, and so this work closer to artificial life than classic artificial intelligence.

In CS0 and the high school components, we use NetLogo[8] (Tisue & Wilensky 2004; Wilensky 2002) to create such simulations. NetLogo is built on top of Java, and provides an environment for writing agent-based programs — every entity in the environment is an agent, and one programs the agents by providing them with a set of actions that they can perform. NetLogo comes with a library of simulation models, including classics such as the Boids (Reynolds 1987) flocking model, and a simple predator-prey scenario (in the case of NetLogo, this involves wolves and sheep. Both these models are depicted in Figure 4.

[7] `http://www.surveyor.com/`

[8] `http://ccl.northwestern.edu/netlogo/`

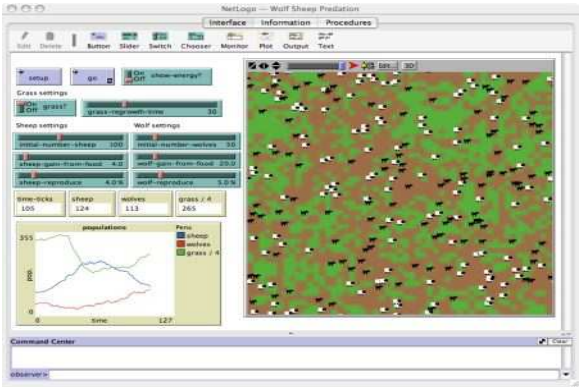


Figure 4: Sample NetLogo simulation: predator-prey

In both these classes, we start by introducing the students to the models, and allowing them to explore the way in which changing parameters (which NetLogo allows one to do while the simulation is running by moving sliders on the interface) alters the behavior of the model. Thus the students learn how the shape of the flock will change, or the balance between predator and prey will alter, as they change the speed with which flocking agents can turn, or the probability that prey animals will reproduce.

Following this exploration period, students are encouraged to create their own models. These are typically ecosystems that bear a strong similarity to the wolf/sheep model[9], though some are more ambitious — a recent example being a route-finder for the New York subway system[10].

In CS1 and CS2, the students write the simulations from scratch in C++, and without the support that NetLogo provides, the results are less impressive simulations. However, we still manage to have the students produce small ecosystem examples with simple rules guiding the behavior of the agents. As with the robotics flavor, we insist that students in CS2 use more advanced programming techniques in developing their simulations than the students in CS1.

Multi-agent Games

Games are an excellent motivational tool for encouraging students at all levels. They provide a method to introduce basic concepts in computer science, programming and artificial intelligence. For creating games we have adopted the Scratch environment. Scratch[11] (Peppler & Kafai 2007) is a graphical programming language, similar to Alice[12] (Cooper, Dann, & Pausch 2003). It allows programmers to create 2D interactive games. The characters or agents in each game are called *sprites*. Sprites can be programmed to sense information, react to the environment and to other sprites, as well as interact with the user. Students cre-

[9] For some reason many of them involve fish.

[10] Examples can be found at `http://bridges.brooklyn.cuny.edu`.

[11] `http://scratch.mit.edu/`

[12] `http://www.alice.org/`

ate scripts or programs for each sprite by stacking together graphic blocks.

In Figure 5 the script for the behavior of the blue fish is displayed in the center column. Blocks of different types can be dragged from the left column and stacked together to create a program. In the FishChomp game (upper right corner), the user interacts with the blue fish by moving the mouse. In order to increase the score and move on to the next level, the blue fish must eat the swimming divers. Each of the divers are also agents with their own programs.

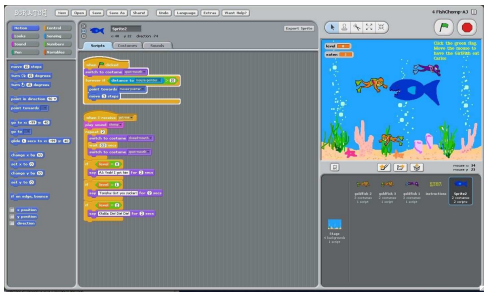


Figure 5: Sample Scratch game

Evaluation

Project evaluation involves collecting and examining two types of data sets. The first data set consists of pre- and post-surveys, administered to all students enrolled in Bridges sections of undergraduate courses and all students who attend high school activities. Following standard IRB[13] procedures, survey completion is optional, and students are provided with information about the Bridges project to help them make informed decisions about whether to complete surveys or not. A random set of Non-Bridges sections of the undergraduate courses have also received surveys.

The purpose of the surveys is primarily to: (1) identify the demographics of the student populations, particularly focusing on gender, language spoken at home, higher education obtained by family members, etc.; and (2) determine if students' perception of the field of computer science, and of computer scientists, changes by participating in interventions that are actively interdisciplinary. The data presented in the following figures summarizes nearly 500 undergraduate and high school students who completed surveys between Fall 2006 and Summer 2007. Figure 6 shows the breakdown between the genders of students surveyed. The high school programs have more than 50% female students. Partly this is attributable to our recruitment strategy, which involves working with school guidance counselors, science and math teachers to target female students with recognized aptitude in problem solving. The undergraduate classes follow the international trend, whereby female students drop out in larger numbers than male students as the major courses advance. We note that the retention rate for female students in CS1 increased between Fall 2006 and Spring 2007, the first two semesters of the Bridges program. Figure 7 shows the wide range of languages spoken at home by students in the Brooklyn College community. A total of 38 dialects are spoken. Just over half (59%) of students speak English at home.

[13]Institutional Review Board

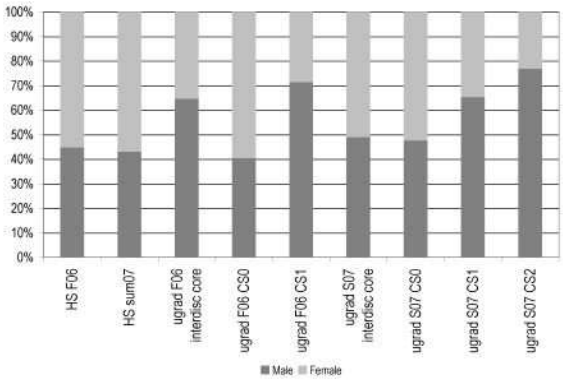


Figure 6: Gender breakdown of students surveyed

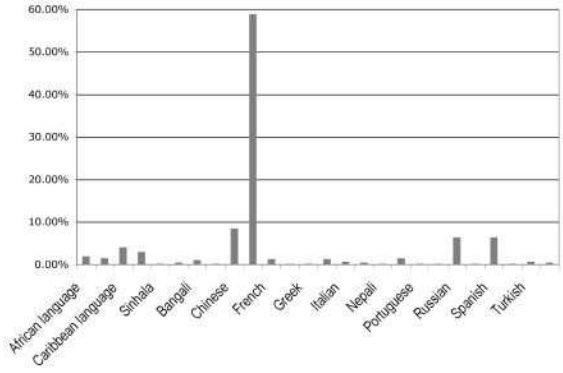


Figure 7: Languages spoken at home by students surveyed

Measuring change in attitude is difficult. We have been experimenting with several types of pre-/post-survey questions in an attempt to capture student's changes in perception of the field of computer science. A post-survey administered in Spring 2007 asks students to describe uses for computer science; these results show that students have gained a broad understanding of CS as an interdisciplinary field, with applications ranging from urban search and rescue robots to medical data mining. Another interesting survey question asked students to list 3 words that describe a computer scientist. We collected data on this question for both pre- and post-surveys in Spring 2007. Table 1 gives some indication of change. Words in the first column of the table that are followed by an asterisk (*) include all words that can be derived from the word stem; for example, "educat*" could be "educated" or "educator". Some words are conglomerates; for example, "smart" encompasses "smart", "intelligent", "brilliant", "genius", etc. It is interesting to note the changes. In

particular, the number of students who indicated that computer scientists are "smart" decreased. This could be either because the instructors made a bad impression and students left their courses thinking that their professors were not so intelligent after all, or — the interpretation we prefer — because after taking the courses, students feel the subject matter is more approachable and therefore one does not have to be a genius to major in computer science. We also note that the number of students who indicated that computer scientists are "geeks" (or "anti-social", etc.) also decreased; one student even went so far in the post-survey as to indicate that computer scientists are cool!

word	*pre*	*post*	*word*	*pre*	*post*
smart	136	112	solv*	9	12
educat*	8	1	patient	43	34
math*	24	16	methodical	1	2
logical	17	24	determined	7	8
program*	10	10	precise	6	7
geek	20	14	creative	27	16
anti-social	3	0	innovative	5	9
cool	0	1	interest*	7	4
boring	5	6	curious	7	4

Table 1: Pre- and Post-survey results: "Write down 3 words that describe a computer scientist", undergrad Spring'07.

The second data set consists of enrollment data. Here we are particularly interested in retention. As illustrated in figure 1, historically, students tend to leave the major in droves after taking CS1, while higher retention is obtained for students who complete CS2. Through a combination of better advising students regarding their placement in CS1 vs CS0 as well as providing a more applied, socially relevant, interdisciplinary and hands-on learning environment for CS1, it is hoped that these numbers will improve. It is too early in the project timeline to be able to determine if the Bridges sections of undergraduate courses have had the desired effect. The first sections were offered in Fall 2006; a comparison with Spring 2007 enrollment data is forthcoming.

Summary

We have described our efforts to broaden the demographic of students participating in computing courses, focusing on the introductory level and bridging students who are underprepared in high school into computer science major courses in college. Our methodology includes a hands-on, cross-disciplinary approach to teaching, with context-based, lab classes at the undergraduate level and after-school programs at the high school level, centering on five applied areas within computer science. We have introduced concepts from artificial intelligence within at least three of these "flavored" areas, in an attempt to engage students early on with problem-solving and understanding that AI is not just the name of a Hollywood movie.

Although the project is not quite two years old, we have already taken a few "lessons learned." These have to do with logistical issues. First, the high school computing preparatory class has had strong enrollments in the Fall, but these dwindle in the Spring as the weather gets warmer and students begin participating in outdoor sports teams and preparing for high-stakes state exams given in late Spring. Second, the undergraduate students tend to register for courses based on what fits into their schedule, not according to which flavor is more interesting to them. As a result, we are considering offering multi-flavored sections, so that students get a sense of computing applied in a broad, interdisciplinary way. Third, teaching computing applied to a particular context works well only if the contextual application can be easily explained—if the instructor has to provide a lot of scaffolding in order for the students to understand the example, how it pertains to computing and/or how a computing solution can be applied, then students lose interest in the subject and do not gain understanding of the underlying computing concepts (let alone their connection to a context). Fourth, hands-on instruction is not necessarily a natural way to teach for all faculty. Some training may be needed in order to adapt such a methodology widely across a department so that instructors understand how to use lab time effectively. Finally, hands-on instruction not only has pedagogical gains, but also social gains—faculty get to know students better and vice versa. Students feel less threatened by faculty and view them as more approachable.

Acknowledgements

This work was supported by NSF BPC #05-40549.

References

Cooper, S.; Dann, W.; and Pausch, R. 2003. Teaching objects-first in introductory computer science. In *Proceedings of SIGCSE*.

Erwin, B.; Cyr, M.; and Rogers, C. B. 2000. LEGO engineer and ROBOLAB: Teaching engineering with LabVIEW from kindergarten to graduate school. *International Journal of Engineering Education.*

Moskal, B. M.; Laisch, D.; and Middleton, N. 2001. Science related degrees: Improving the retention of women and minorities through research experience, mentoring and financial assistance. In *Proceedings of the ASEE Conference*.

Peppler, K., and Kafai, Y. B. 2007. From supergoo to scratch: Exploring media creative production in an informal learning environment. *Journal on Learning, Media, and Technology* 149–166.

Reynolds, C. W. 1987. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics* 21(4):25–34.

Sleeter, C. E., and Grant, C. A. 1987. An analysis of multicultural education in the USA. *Harvard Educational Review* 57:421–444.

Tisue, S., and Wilensky, U. 2004. Netlogo: A simple environment for modeling complexity. In *Proceedings of the International Conference on Complex Systems*.

Wilensky, U. 2002. Modeling nature's emergent patterns with multi-agent languages. In *Proceedings of EuroLogo*.