Building Worlds: Bridging Imperative-First and Object-Oriented Programming in CS1–CS2

Zoë Wood
Computer Science Department
California Polytechnic State University
1 Grand Avenue
San Luis Obispo, CA 93407 USA
zwood@calpoly.edu

Aaron Keen
Computer Science Department
California Polytechnic State University
1 Grand Avenue
San Luis Obispo, CA 93407 USA
akeen@calpoly.edu

ABSTRACT

When teaching introductory computing courses, we are often guilty of writing rudimentary programming assignments – those meant to illustrate one simple language feature, comprised mostly of code that will never be used beyond the assignment. Admittedly, first-year computing students must navigate a myriad of challenges, sometimes learning both imperative and object-oriented programming, in addition to mastering syntax, logic, debugging, and testing. To tackle the difficulties of developing CS 1 and CS 2 courses that engage students in learning while addressing the numerous course objectives, we chose to challenge students to create virtual worlds in one large comprehensive two-quarter long programming project. Students were granted creative freedom within a framework that gradually introduces many programming skills and that requires the mastery of object-oriented programming and some engaging algorithms. We present the curriculum, performance comparisons, and observations. Overall, we consider the experimental courses a success that will have an impact on our department's future curricular offerings.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*

General Terms

Experimentation

Keywords

Introductory programming, object-oriented programming

1. INTRODUCTION

Creating meaningful learning experiences for students in introductory computing courses is a challenge. Many students find themselves overwhelmed with simultaneously learning syntax, logic, program design, decomposition, debugging, algorithms, testing, and object-oriented design (all of which may be required in a first year assignment). In order to address this overwhelming problem space, faculty often rely on small programming assignments to introduce discrete language or programming concepts one at a time. These rudimentary assignments are problematic as they are often decoupled from real-world programming and real-world problems, lessening student engagement and leading to confusion about the true use of these language/design features. This is especially true for object-oriented programming where the design goals (and language features supporting these goals) run almost directly orthogonal to the context of a small, rudimentary program. In short, language support for programming large systems is often unappreciated and poorly motivated in small programs.

Beginning in 2010 our department tackled some of the challenges of introducing students to computing via the use of project-based contextualized learning in CS 0 [11]. Students are introduced to computing via topics such as mobile computing, video games, music, computational art, robotics, and security. The success of these CS 0 offerings inspired us to consider some of the core challenges we face in the subsequent first-year courses (CS 1 and CS 2). Specifically, we set out to design courses with high student engagement that expose students to large "real-world" problems, that require working on a larger-than-typical code base where each assignment's code is dependent on a prior assignment, and that require the creation of a project that truly illustrates the benefits of object-oriented programming. We present details of these courses in Section 3.

Our envisioned CS 1 and CS 2 courses (referenced from here on as CS $1_{\texttt{ex}}$ and CS $2_{\texttt{ex}}$ to avoid confusion with the prior CS 1 and CS 2) differ significantly from the existing courses in our department. Details of existing courses are provided here for context.

Our department's CS 1 could be said to implement what the Computing Curricula 2001 [1] terms an "imperative-first" approach. Specifically, CS 1 focuses on programming skills using primitive and structured data, expressions, conditionals, loops, arrays, and (non-recursive) functions. Our department's CS 1 uses `C` as the implementation language, but without any required instruction on pointers (aside from incidental uses inherent in `C`). Our department's CS 2 covers objects and object-oriented programming in Java including classes, interfaces, polymorphism, and inheritance. CS 2 is

SIGCSE'15, March 04–07, 2015, Kansas City, MO, USA.

ACM 978-1-4503-2966-8/15/03 ...$15.00.
http://dx.doi.org/10.1145/2676723.2677249.

also tasked with introducing recursion, searching, sorting, and linked lists. A typical instance of this course requires that students implement very prescriptive assignments with an emphasis on "how" the object-oriented language features work in Java.

Though our department's CS 1 builds a sufficiently solid imperative foundation that supports learning to program in Java, many students at the end of CS 2 are left unsure of the need for or value of object-oriented features. In particular, when early CS 2 assignments require the use of interfaces, it is a challenge to properly motivate the reasons for their use. When early CS 2 assignments *do not* require the use of interfaces, it can be a challenge to motivate their introduction later. Some students have questioned the value of object-oriented programming as a whole since they feel they could implement the assignment more directly in `C`; this is typically due to limited additional complexity in the prescriptive assignments that focus on introducing object-oriented features.

We sought to address these shortcomings by bridging the transition from CS 1 to CS 2 with a single project that spans these two courses. We offered one section of an experimental CS 1 (CS 1_{ex}) during the Winter quarter 2014 and one section of an experimental CS 2 (CS 2_{ex}) course during the Spring quarter 2014, each with approximately 35 students. These sections were co-taught by the authors during both quarters. The course-spanning project required students to create their own virtual worlds with interacting entities/characters. Students were allowed a great deal of creativity in designing the theme, the environment, and the behavior of their worlds and their worlds' inhabitants. The instructors provided high-level requirements that drove the development of a larger program while gradually introducing object-oriented features.

During this project, the students had to contend with code dependencies between assignments, rewrite portions of their program to address shortcomings in design decisions (and, ultimately, to switch languages), test modules before integrating them into the larger program, and implement context-based graph searching algorithms (an extension in scope beyond the searching of sequential structures required in our department's CS 1 and CS 2). These experiences pushed student learning beyond the development of small throw-away programs and contextualized the value and need for more advanced language features and design methodologies.

2. RELATED WORK

There is a large body of work addressing the multitude of challenges (including failure rates [5]) in introductory computer science curricula, including context-based computing, small courses and the use of labs early, making use of pair programming, etc. [10, 8, 13, 7, 6]. Our department has a strong commitment to introductory computing: the introductory courses are kept relatively small with about 35 students per section, each section includes equal lab and lecture time, and the department runs a free tutoring center. Though these academic structures are beneficial, students in introductory courses continue to fail at higher than acceptable rates and there is a perceived mismatch between the skills students gain in introductory courses and the skills desired by upper-division instructors.

In order to address issues related to student failure rates, dwindling enthusiasm, language battles, and upper-division skill mismatch, we proposed the experimental course structure described in this paper. The work presented here is strongly founded in the success of using context-based computing in introductory computing courses [10, 8, 11] and our belief in the use of project-based learning [4], especially when considering teaching the utility of object-oriented language features. Additionally, we were inspired to allow students to work in pairs for all major programming assignments, using a casual implementation of pair-programming [13]. Finally, our explorations into using Python as a first programming language were influenced by various sources suggesting Python as a positive starting language [14, 9].

Though our proposed context of a two-dimensional virtual world to motivate learning object-oriented language features may appear reminiscent of GridWorld [12], our work was inspired independently. Moreover, our project requires students to create their own worlds and to write all associated code from scratch. This structure provides a means for students to experience the ramifications of non-ideal design decisions and to appreciate the value of improved design in addressing the shortcomings of earlier decisions.

3. OUR SOLUTION

To address the challenge of developing a CS 1 and CS 2 course sequence that engages students in learning while allowing them to apply their learning to one large comprehensive programming project, we designed a two-quarter sequence focused on the construction of a virtual world[1]. Students were granted creative freedom within the framework of the required technology specifications.

Initially, the students were asked to design worlds that met the following requirements:

- The world is confined to two dimensions.
- The world must contain multiple characters with different behavior. In particular, the first specification of the world required the definition of:
  - a **gatherer(s)** that would move through the world seeking a resource.
  - a **generator(s)** that would generate a resource (in a location relative to the generator).
- The world must include consumable resources that are *generated* and *gathered*.

Later extensions to the project required:

- Obstacles in the world that prevented or restricted movement.
- Character transformations based on interactions with the world, with a resource, or with another character.
- A world changing event affecting all elements of the world within a given (geometric) range.

[1]Those interested can find materials for this curriculum at `http://users.csc.calpoly.edu/~akeen/courses/csc101x`.

Although it was not necessary, we opted to require that each world be displayed in 2D in order to give the students improved visual feedback on the state of their world and to allow for creative visual expression to enhance engagement.

In order to address the diverse curricular needs of CS 1_{ex} and CS 2_{ex}, each quarter had a slightly divergent emphasis and development environment.

- CS 1_{ex}: The first course focused on programming skills using primitive and structured data (via classes and objects), expressions, conditionals, loops, lists, and functions. This course used Python as the implementation language with worlds displayed using pygame [3].
- CS 2_{ex}: The second course focused on object-oriented programming with increased exposure to classes and objects and an introduction to interfaces, polymorphism, and inheritance. This course also covered recursion, searching, sorting, and the implementation of linked lists. Due to our department's requirement that students entering CS 3 must know how to program in Java, this course included a transition from Python and pygame to Java with Processing [2].

Lecture material covered introductory programming and project-relevant topics with students completing short related lab assignments each week and with the virtual world project split into multiple major homework assignments. Students were allowed to choose between working on these major assignments in pairs or as individuals. Over the two quarters of this experiment, the composition of pairs and individual projects varied but at the end of CS 2_{ex} eleven projects were completed by pairs and eight were completed by individuals. Key assignments were as follows:

- CS 1_{ex}, HW 1: Provide a description of the theme, environment, and characters in the virtual world. Define data representations for some of the characters in the virtual world.

  This assignment introduced the entire virtual world project by requiring students to describe the world they intended to build. This assignment also introduced students to programming and Python through the implementation and testing of simple classes and objects to represent characters in the world (as defined by properties such as the x-/y-location, the number of resources gathered, and the frequency at which a resource is generated). These two extremes provided a creative context (the grand vision) in which to project the introductory minutiae of defining simple data (the characters).
- CS 1_{ex}, HW 2: Build the basic structure to visualize the world in 2D. Populate the world with characters and implement movement for a subset of them.

  This assignment required that a gatherer move toward a resource and stop once the resource is reached. In this context, the students were introduced to decision-making, conditional logic, and looping. This assignment also served to introduce pygame for displaying the world.
- CS 1_{ex}, HW 3: Increase the complexity of interactions in the world, increase the visual complexity of the world, and visualize reactions to interactions.

  This assignment required support for primitive user-interaction via pygame events (both mouse and keyboard) including a visual response to selection of resources and movement of a gatherer to the nearest or farthest resource (by choice). In addition, the visual theme of the virtual world was improved through the selection or creation of sprites.
- CS 1_{ex}, HW 4: Build a program to allow a user to "create" a specific instance of a world by placing entities and background sprites using the mouse and keyboard (including functionality to save and load worlds).

  This assignment required the management of multiple characters (objects) of varying types through the use of lists and the 2D grid. This included the presentation of characters that could be placed over different types of background tiles necessitating managing multiple objects in the same 2D world "cell". This assignment also emphasized data conversion in the form of save files (conversion of an object to/from a string) and movement of a "view window" in the world (conversion of window coordinates to/from world coordinates).
- CS 1_{ex}, HW 5: Bring the world to life by implementing clock-based movement and animation. Some characters must transform (with a corresponding animation) based on actions taken.

  This assignment completed the first phase of the two-quarter virtual world project with a "live" world in which characters move through the world to take actions (e.g., gather or deposit resources). This required triggering actions based on a timer, preventing movement of characters through obstacles (with some naive effort to bypass obstacles), and changing the state of the world (creating and/or removing entities) during execution.

Though it was infeasible to force students completing CS 1_{ex} to continue with CS 2_{ex}, of the eleven students in CS 1_{ex} that did not continue on to CS 2_{ex}, none continued on to any CS 2. The majority of these students were "non-majors" that are not required to take CS 1 or CS 2.

While we did have control over registration priority for CS 2_{ex}, enrollment demands necessitated adding ten students to CS 2_{ex} that had not taken CS 1_{ex}. These ten students had completed traditional non-objects CS 1 courses (using `C`). To support these new students and to aid any continuing CS 1_{ex} students that did not complete their assignments perfectly, we offered the opportunity to start fresh with an instructor provided world implementation (approximately 1700 lines of code). Five projects were continued from the students' own original code, eleven projects used the instructor provided code, and three projects started with the instructor code but modified it to fit their own world. Major assignments in CS 2_{ex} included:

- CS 2_{ex}, HW 1: Refactor the code to group functionality for each character/class into methods. Justify this factoring in a written design document.

  This assignment required moving the reasonable associated behavior for each character (and other class) from stand-alone functions (as used in CS 1_{ex}) to methods. Through this assignment students were introduced to objects with behavior, extending their notion of a data-holding object from CS 1_{ex}.

- CS 2_{ex}, HW 2: Improve character movement within the world.

  This assignment required improving character movement by replacing the naive search from CS 1_{ex} with the A* (A-star) search algorithm. The implementation of A* for this assignment required students to consider the representation of the search space in order to select neighbors, the representation and manipulation of open and closed sets, and how to reinitiate a search (the world remains active, so a character will move again). This assignment also introduced recursion for reconstructing the shortest path.

- CS 2_{ex}, HW 3: Refactor the world code to use inheritance for the world entities.

  In many cases, the movement of stand-alone functions to methods in HW 1 involved duplication of code. In some cases, the duplicated code contained an "instance of" check to determine the action to take. This assignment introduced the concept of inheritance and required that students refactor duplicate code into a parent class while specializing behavior in subclasses.

- CS 2_{ex}, HW 4: Convert the world project from Python with pygame to Java with Processing.

  This assignment introduced Java through a code rewrite. The Python (with pygame) implementation had to be converted to Java (with Processing). Of particular note, this rewrite required that students contend with type annotations, access modifiers, interfaces, checked exceptions, generics, and support for lists in Java's API.

- CS 2_{ex}, HW 5: Integrate a 'world changing' event into the world.

  This assignment required the implementation of a triggered event that starts at a specified location and that has some local extent. Furthermore, the assignment required that the event be visualized by changing the images of the associated grid elements and that the event affect different characters differently (creating both a change in the physical appearance of the characters along with a change in the behavior of one of the character types).

In addition to submitting these assignments, students were required (twice each quarter) to demonstrate their world and code in front of the instructors and their classmates. These class code reviews provided an excellent opportunity for students to share their solutions and discuss their design decisions.

An important note about this experimental course is that it was a break from the traditional CS 1 and CS 2 for our department in many ways. We used different programming languages, different assignments, and a different lecturing style. Both authors worked collaboratively on all aspects of the course: co-designing lab and homework assignments, co-writing exams (lab and lecture), and sharing lecturing (trading-off approximately weekly or by topic). Each instructor attended every lecture and genuinely co-taught the course.

Table 1: CS 1 Traditional vs Experimental scores

	CS 1		CS 1_{ex}	
	30 Overall 28 Computing*		31 Overall 23 Computing*	
	mean	std. dev.	mean	std. dev.
Midterms				
Exam 1				
Overall	80.9%	14.6	82.5%	13.4
Computing*	80.4%	14.8	87.2%	9.0
Exam 2				
Overall	76.9%	15.5	75.1%	23.3
Computing*	76.5%	16.0	83.2%	16.4
Lab Exam				
Overall	82.0%	17.7	82.1%	23.3
Computing*	81.3%	18.0	86.2%	22.2
Final Exam				
Overall	79.7%	14.2	77.4%	17.4
Computing*	79.1%	14.4	82.9%	12.7

4. PERFORMANCE COMPARISONS

In measuring the success of our experimental CS 1_{ex} and CS 2_{ex} courses we consider both student outcomes and the participating faculty's perspectives.

For student outcomes, we compare student success in the experimental section versus the traditional CS 1 and CS 2 courses taught during the same quarters. At this time we do not have statistics on the long-term contribution to student success (in subsequent courses), but can report positive outcomes for each comparative section.

We compare the performance in the CS 1_{ex} course against a traditional CS 1 course taught by one of the authors during the same quarter. The same midterms, lab exams, and final exam were used in each course, but with each adapted to the appropriate syntax for the course (`C` in CS 1 and Python in CS 1_{ex}). Table 1 lists each section's scores for students that took the final exam, broken down into the 'Overall' class and the 'Computing*' students, which includes computer science, software engineering, computer engineering, and 'liberal arts and engineering studies' (LAES) majors (these are the majors that typically continue on to CS 2). When comparing all students ('Overall') in each course, the CS 1_{ex} students performed slightly better or just barely below their their traditional CS 1 counterparts in all assessments.

It is worth noting that the experimental section student population included an uncommonly high number of non-computing majors. The creative theme attracted ten non-computing and non-engineering majors (including students from art, business, political science, and agriculture) out of a total student population of 36 students, while in the traditional class the vast majority of students were computing or engineering majors. When comparing computing and engineering majors between the two sections(designated 'Computing*' in Table 1), *the students in the experimental section did better in all categories.*

For student outcomes in CS 2_{ex}, we can only compare the experimental section against traditional sections taught by faculty members *other* than the authors. In addition, due to the more divergent material in the experimental CS 2_{ex} course (with its heavy emphasis on path-finding algorithms

and object-oriented design/refactoring from an existing large code base), the material between the CS 2 variants was more diverse. However, the same final exam was used across experimental and traditional sections with the exception of two questions out of twelve (20 points out the the 95 total points). For the experimental section the average on the final exam was 80.8%, while two other traditional section final exam averages were 79.6% and 83%. Comparing the total number of students enrolled in the CS 2 sections (experimental versus traditional), 82% of the students in the experimental section received a passing grade, while 86% of the students from the two traditional sections received passing grades.

In total, the performance comparisons indicate that the students in the experimental section performed on par with their counterparts in the traditional sections. We feel students in the experimental sections had additional gains documented below.

5. OBSERVATIONS AND IMPRESSIONS

When reflecting on the CS 1_{ex} and CS 2_{ex} experiment, the participating faculty perceived some very clear positive outcomes. Perhaps the clearest outcome for both faculty was an increased appreciation for the use of Python as an introductory programming language. Using Python in CS 1_{ex} drastically reduced the amount of lab and office hour time that instructors spent explaining strange and obscure program errors so common with `C` in an introductory course. We will not repeat all of the advantages here, as many are well-documented [14, 9], but, for example, beginning with Python instills some simple good programming practices from the outset, such as initializing variables before use.

Moreover, the transition from Python to Java in CS 2_{ex} provided a means to bootstrap object-oriented concepts in a manner that is not easily replicated when transitioning from `C` to Java. In particular, since CS 1_{ex} introduced objects as aggregates of data, we could begin CS 2_{ex} with a discussion of adding behavior to objects (still in Python) without any concern for the syntactic and semantic overhead of access modifiers and `static` data/methods. Continuing in Python, the introduction of inheritance was simplified by the omission of access modifiers and their interaction with inheritance and by the omission of the distinctions (in Java) between extending a class and implementing interfaces. In addition, each modification (e.g., the refactoring of a function into methods or joining a set of classes at a parent through inheritance) could be made and tested iteratively since the function-based aspects of the code continue to work in Python. After this experience, students found these concepts (objects with behavior, inheritance, polymorphism) more readily understandable in Java and showed greater appreciation for the use and value of interfaces.

Another aspect of the experimental offering that we found to be very effective was the use of a large, cohesive project in CS 2_{ex}. The large virtual world programming project requiring complex interacting objects with some shared and divergent behavior (benefiting from polymorphism and inheritance), made the introduction of many of the object-oriented topics more natural and immediately useful to students. The 2D world context provided a great stage for students to apply searching/pathing algorithms; the introduction of these algorithms improved student engagement leading to many in-class discussions and outside research into and coding of variants of the pathing algorithms. Even students that struggled with this topic continued to be engaged and to work hard on these projects in order to accomplish their goal of moving their gatherer to a resource along a "good" path. For some, this led to continued work on tuning their pathing during later assignments when the emphasis was no longer on pathing.

Some of the general observations we came away from this experience with are as follows. In the experimental sections, it was a positive experience for faculty and students:

- to include project appropriate pathing algorithms in CS 2_{ex} (specifically depth first search and A*).
- to have assignments that built on prior assignments (students were more accountable for their code and had to "stick with it" to get their program to work).
- to rewrite a large Python code base in Java. This activity required a level of precision and code understanding that previously used rudimentary assignments did not require.
- to use object-oriented language features in the context of a relatively large code base to implement the virtual world.

In addition, we strongly suspect that the students in our experimental section have a better understanding of the need for and value of object-oriented language features and object-oriented programming in Java than they would have after completing our traditional CS 2 (though our CS 2_{ex} students were not exposed to as much of the Java API). We intend to do longitudinal tracking for these students in CS 3 to evaluate their performance compared to students in traditional sections.

The aspect of the experimental section that was most challenging for students (reflected in their verbal comments on the class) was requiring two different programming languages in one quarter (during CS 2_{ex}). However, each student had a diverse reaction to the two very different languages they needed to use throughout this two-quarter long project. At the end of CS 2_{ex} we asked students to express their preference for Python or Java (including the option of saying they liked both). Twelve students commented that they enjoyed Java while eighteen said that they did not like Java or that they preferred Python. Student comments about the languages included opinions such as: "I hate shoving everything into classes", "I don't like types", "I like the freedom of Python", "Java had a great IDE", and "I liked the constraints and structure in Java". This varied opinion of the two languages is a positive indicator that exposing introductory students to various languages allows them to find the programming tools that best match either the specific problem they are solving or, when given a choice, that best match their own computational thinking style.

The one aspect of the experimental sections that did not go as well as predicted was the very open-ended nature of the projects at the beginning of CS 1_{ex}. We found that during the first few weeks of CS 1_{ex}, when students were designing their worlds but had fairly limited programming experience, they struggled a bit with understanding what they needed to do and what they *could* accomplish. For future offerings of this sequence, we strongly feel that providing more structured assignments while introducing world designing and building would be beneficial in CS 1.

Figure 1: Example world visualizations developed during CS 1_{ex} and CS 2_{ex}: base world provided post-CS 1_{ex}(left); a world with knights gathering diamonds (middle); a world featuring witches gathering potion ingredients (right).

Similarly, the one aspect of the experimental sections that was disappointing for the participating faculty was the fact that allowing students the creative freedom to design their own worlds was less important to the students than expected. Gauging student interest based on those who continued with some version of their own world in CS 2_{ex}, only 42% of the students were committed to designing their own individualized creative world. This commitment to an individualized world did vary by student ranging from those that clearly made little effort to find a set of visually cohesive sprites to those that took the time to create their own set of sprites. Some examples of virtuals worlds are shown in Figure 1. These examples include a world where knights seek diamonds and a world where witches gather potions. A particularly notable student-designed world modeled molecules joining together to create more complex structures.

6. CONCLUSIONS

Overall, we consider the experimental courses a success that will have an impact on our department's future curricular offerings. Very few aspects did not work as well as predicted while other aspects like language and context-based object-oriented projects were very strong wins. One strong outcome from this experiment is the department voting to change the programming language used in CS 1 from `C` to Python in all sections. Longer term curricular outcomes include examining the use of a quarter-long project-based curriculum in CS 2 and an examination of the topics and sequence of topics in CS 2 and CS 3.

The experience, in and of itself, of co-teaching this course is worth commenting on. The process was very rewarding but not without its challenges in terms of the overhead in communication and variation in teaching style. As university professors we so often are alone in our classrooms and do not need to share the stage. The process of seeing another faculty member teach a co-designed lesson is incredibly rewarding – providing insight into methodology for explaining certain concepts and insight into general classroom and lecturing style. This experience encouraged one of the instructors, with a typically reserved teaching style, to experiment with different forms of in-lecture interaction.

7. REFERENCES

[1] Computing curricula 2001. *J. Educ. Resour. Comput.*, 1(3es), Sept. 2001.

[2] Processing– http:\\www.processing.org, 2014.

[3] pygame– http:\\www.pygame.org, 2014.

[4] M. Barg, A. Fekete, T. Greening, O. Holl, J. Kay, H. Kingston, and K. Crawford. Problem-based learning for foundation computer science courses. *Computer Science Education*, 10:109–128, 2000.

[5] J. Bennedsen and M. E. Caspersen. Failure rates in introductory programming. *SIGCSE Bull. 39*, pages 32–36, 2007.

[6] K. E. Boyer, R. S. Dwight, C. S. Miller, C. D. Raubenheimer, M. F. Stallmann, and M. A. Vouk. A case for smaller class size with integrated lab for introductory computer science. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '07, pages 341–345, New York, NY, USA, 2007. ACM.

[7] J. P. Cohoon and L. A. Tychonievich. Analysis of a CS1 approach for attracting diverse and inexperienced students to computing majors. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 165–170, New York, NY, USA, 2011. ACM.

[8] Z. Dodds, R. Libeskind-Hadas, C. Alvarado, and G. Kuenning. Evaluating a breadth-first CS 1 for scientists. In *ACM SIGCSE Bulletin*, volume 40, pages 266–270. ACM, 2008.

[9] M. H. Goldwasser and D. Letscher. Teaching an object-oriented CS1 – with python. *SIGCSE Bull.*, 40(3):42–46, June 2008.

[10] M. Guzdial. Does contextualized computing education help? *ACM Inroads*, 1(4):4–6, Dec. 2010.

[11] M. Haungs, C. Clark, J. Clements, and D. Janzen. Improving first-year success and retention through interest-based CS0 courses. In *Proceedings of the ACM Technical Symposium on Computer Science Education*, 2012.

[12] C. Horstmann. Gridworld– http:\\horstmann.com\gridworld, 2008.

[13] N. Nagappan, L. Williams, M. Ferzli, E. Wiebe, K. Yang, C. Miller, and S. Balik. Improving the CS1 experience with pair programming. *SIGCSE Bull.*, 35(1):359–362, Jan. 2003.

[14] A. Radenski. "Python first": A lab-based digital introduction to computer science. *SIGCSE Bull.*, 38(3):197–201, June 2006.