Using Python and QuickDraw to Foster Student Engagement in CS1

Ben Stephenson

Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
ben.stephenson@ucalgary.ca

Abstract

Most students enrolling in introductory computer science courses today have grown up working exclusively with graphical user interfaces. Through this experience, they have come to expect computers to be reasonably intuitive and easy to use. Yet, when students enter an introductory course in computer science, it is commonplace for them to work on exclusively text based assignments using complex programming languages.

This paper describes our experience using Python and QuickDraw when teaching CS1. We have found that using these tools engages students by allowing them to quickly complete assignments that solve interesting problems and generate visually interesting results, without requiring excessive exposure to low level concepts or the use of complex graphical libraries. In our experience, students get greater satisfaction from working on such assignments, and this encourages them to spend additional time and effort learning the concepts that each assignment is designed to reinforce. In addition, using these tools presents a modern image for the discipline while also providing opportunities for students to be creative, helping to overcome the stereotype that computer science is sterile and boring.

Categories and Subject Descriptors K.3.2 [*Computers and Education*]: Computer and Information Science Education—Computer Science Education

General Terms Algorithms, Design, Languages

Keywords CS1, Python, Pedagogy, Computer Science Education, Graphical Projects, Student Engagement

OOPSLA 2009, October 25–29, 2009, Orlando, Florida, USA.

1. Introduction

We have taught CS1 and CS2 on several occasions over the past 5 years. During that time, we have been asked to use several different languages including Pascal, Java and C++. Each of these languages has specific advantages and drawbacks when used in an introductory course which have been examined in detail by other educators [12]. Of these languages, we considered Pascal to be a good teaching language because it was relatively easy to use while still supporting detailed discussions of value vs. reference parameters, and pointers. More recently, we have found that many students prefer the use of Java or C++ because they perceive these languages as being directly applicable to their future career. An analysis of languages requested in employment advertisements revealed that they are the two most frequently requested languages [5], suggesting that student perceptions are accurate in this regard. C++, Java and Pascal are all statically typed languages which typically require source code to be compiled before the program is run. As such, variables must be declared before they are used and the programmer must explicitly compile their program after each change.

This year, we have restructured CS1 so that it is taught using Python. Python is a strongly, but dynamically, typed language that is normally executed using an interpreter. As such, it permits rapid development and experimentation by removing many of the restrictions of a static type system while also allowing students to test changes to their programs without a visible compilation phase. In our experience, students also see Python as a relevant language to learn due to its use by high profile companies such as Industrial Light and Magic, and in successful games such as Sid Meier's Civilization IV.

In addition to introducing Python, we have also introduced QuickDraw [19] into our CS1 class. QuickDraw is a language and platform independent multimedia package that provides a diverse collection of operations including two dimensional and three dimensional graphics primitives, mouse input, and audio file playback. QuickDraw reads commands from standard input, generating the graphical or audible re-

sults of those commands as output. As such, student's programs can generate graphical output by printing QuickDraw commands to standard output, and then piping this output to QuickDraw, which will display the appropriate results as the student's program runs. Several other solutions have been proposed for introducing graphics into introductory classes without the complexity of using DirectX or OpenGL directly [4, 9, 17, 18, 20]. We elected to work with QuickDraw because of its platform independence, and because we can continue to use it when C++ is introduced in CS2. However, we suspect that the successes we have had could also be realized with other tools that provide a simplified graphics environment within Python.

This paper describes our experience with transforming our CS1 course from text-based assignments in more traditional languages such as Pascal, Java and C++ to assignments with graphical aspects developed in Python. As expected, we observed a significant increase in student interest and engagement. However, we also encountered a number of relatively minor, but unexpected, pitfalls. Our successes, and our techniques for overcoming the unexpected challenges, are discussed in the remainder of this paper.

2. Strengths of Using Python and QuickDraw

Our primary reason for introducing QuickDraw and Python into our first course in computer science was to engage and motivate students while still teaching important introductory computer science and programming concepts. Like many other institutions across North America, we have seen a significant decrease in enrollment over the past few years [1, 6, 14, 16]. It was our hope that tackling larger, graphical, problems with fewer low level details would help excite our students about computer science, and promote increased interest in subsequent courses. We believe that we have been successful in our goal as we have noticed a much greater level of enthusiasm in our students since the change was made. In particular, we observed that students found graphical problems particularly engaging, as described in further detail in Section 2.4. However, additional benefits were also achieved which are described in the following sections.

2.1 Simple Programs are Simple

We have previously taught introductory computer science using Pascal, Java and C++. In each of these languages, the programmer is required to provide some supporting code outside of the body of the main function / method. In some cases, the amount of supporting code that is required is minimal, and only involves language structures that students will quickly become familiar with. For example, in Pascal, it is necessary to wrap the main program in a begin/end pair. While we find the relative simplicity of Pascal appealing for teaching, students expressed significant dismay with being taught what they perceived to be a "dead" language. As a result, we believe that the image problem associated with using Pascal far outweighs its benefits when we are trying to present a modern image that will attract and retain students.

```
class HelloWorld
{
  public static void main(String args[])
  {
    System.out.println("Hello World!");
  }
}
```

Figure 1. "Hello World" implemented using Java

```
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello World!" << endl;

  return 0;
}
```

Figure 2. "Hello World" implemented using C++

```
print "Hello World!"
```

Figure 3. "Hello World" implemented using Python

Using Java or C++ introduces significantly more complexity. For example, Java requires the programmer to declare a class containing a public static main method before one can even consider solving a problem. C++ is similarly problematic, requiring the programmer to include a header file to support IO, to declare a main function, and to either specifically indicate use of the standard name space or fully qualify the name of the output stream. Furthermore, some C++ compilers will generate an error if one fails to have an integer return type for main while not actually requiring that a value is returned.

In contrast to Java and C++, a simple Python program is truly simple. The source code for "Hello World!" programs written in Java, C++ and Python are shown in Figures 1, 2 and 3 respectively. These figures show that while only a single line of code is required in Python, writing this program in Java requires 7 lines of code which introduce two different scopes, a method that takes an array parameter, and the invocation of a method on a data member in a class. The C++ program is also highly complex, as outlined previously. In our experience, students do not understand the complexity of what they are doing. They simply view all of the code outside of the body of the main program as a complex "magical incantation" that is necessary before they

can even begin to consider creating a program to solve the simplest of problems. However, because the supporting code is complex, a single character error can result in cryptic error messages that many students are unable to decipher.

Python does not suffer from this problem. Because a simple program only consists of statements that the students write themselves, they have a much greater understanding of the intended purpose of those statements. As a result, they are much more able to resolve their own problems by carefully and logically examining their code. Other authors have compared the complexity of simple Python and Java programs in greater detail [11], finding that students must master twice as many non-obvious ideas to create a simple Java program compared to its Python equivalent.

The simplicity of Python extends from its source code into the domain of running the program. In order to run the Java program above, one must first compile it using `javac`, creating the file `HelloWorld.class`. Then one must invoke the Java virtual machine to run the resulting class file. Within this process there is a confusing asymmetry where one must include the `.java` file name extension when invoking `javac`, but must not include the `.class` file name extension when invoking the Java virtual machine. In contrast, a Python program can be run using the Python interpreter without explicitly compiling it first, allowing new programmers to enter a single command to run their program after modifying it.

The simplicity continues as variables are introduced and input is read from the user. Python does not require programmers to declare variables before they are used – any time a value is assigned to a previously unused name a new variable is created with that name. Python also provides a simple input function that allows students to write their first programs involving user input with minimal difficulty. In contrast, simple numeric problems such as writing a program that adds numbers, or that performs a simple conversion between units of measure, must be preceded by a discussion of data types in the languages that we have used previously. This adds even more details that the students must consider while they are still struggling to master the basic elements of the language's syntax.

We continue to leverage Python's relative simplicity as the course progresses. Previous work has shown that solving a problem with Python often requires less programmer effort than solving the same problem using C++ or Java [15]. This provides our students with extra time compared to previous versions of the course, allowing them to tackle larger problems.

2.2 Good Indenting is Required

Over the years, we have seen students use a variety of styles of indenting. These have varied from beautifully indented code which clearly emphasizes where each block begins and ends to code devoid of any indenting whatsoever, where every statement is aligned with the left margin. We have also observed numerous "creative" indenting styles, some of which were well intentioned but showed a lack understanding, others of which have reflected insufficient effort on the part of the student.

In Python, white space is of syntactic importance. In particular, the bodies of if statements, loops and functions are all determined by examining the level of indenting of subsequent statements. As a result, students are unable to write correct code without reasonable indenting. Furthermore, Python will generate clear error messages when a line of code is not indented at a recognized level, providing students with immediate guidance on how to indent their code correctly.

2.3 Experimenting is Easy

Python allows students to start creating programs more quickly and easily than with the languages that we have traditionally used. In addition to being easy to use, Python also provides an easy environment for students to experiment with new concepts.

When Python is started in interactive mode, Python statements can be entered directly from the keyboard. Each statement is evaluated as it is entered, and new statements can refer to variables or functions created by previous statements. In addition, if a Python expression is entered which evaluates to a result, it is evaluated immediately and the result of the expression is displayed without the need for a separate print statement. This provides students with an extremely fast and easy to use mechanism to experiment with the new language that they are learning.

QuickDraw also provides a similarly easy mechanism for experimentation. Because it reads commands from standard input, students can also start QuickDraw interactively. QuickDraw commands can be entered directly from the keyboard and the result can be observed immediately. Interacting with QuickDraw in this manner helps students to become familiar with graphical concepts quickly. We have generally begun by introducing concepts such as the graphics coordinate system, mixing red, green and blue components to form new colors, and placing two dimensional primitives. As the course progresses, we also examine concepts related naming and moving primitives. This more advanced topic can also be experimented with by running QuickDraw interactively.

We strongly believe that using Python and QuickDraw interactively helps students to quickly become familiar with the syntax of Python statements and QuickDraw commands. Using these tools interactively provides immediate feedback, and eliminates the overhead associated with saving the file, compiling it, and running the program again. As experienced programmers who have performed thousands, or perhaps millions, of these cycles, the removal of this process may seem like a nice, but minor, convenience. However, the majority of the students entering our CS1 classes have never used a command prompt or terminal window to enter commands before. We believe that allowing students to interact

with the tools that they will use throughout the course in a manner that minimizes the frustration of mis-typed commands or forgotten compiles is a valuable step in building the confidence and sense of accomplishment that will allow them to succeed when facing larger problems.

2.4 Graphical Assignments are Engaging

In the past, we have used exclusively text based assignments in CS1 and CS2 because the complexity associated with commercial graphical libraries was simply too large for students to handle at this level. While several libraries have been developed specifically for educational use, these libraries require a substantial investment on the part of the instructor and teaching assistants to become experts on debugging all of the creative, incorrect ways that students can abuse even the easiest libraries to use. Given the number of different languages that we have been asked to use over the past 5 years, we had significant concerns that a major investment of time into a language-specific graphics package would be immediately followed by a change of language, and as such, a change of library. Using QuickDraw has allowed us to minimize the losses that result due to a change in language while still having an appropriate level of complexity for use in CS1.

We have used exclusively graphical assignments in CS1 for the past two terms, including our first assignment which students complete before being exposed to decision making constructs. In the current term, our first assignment asked students to create a program that is capable of drawing some sort of a face at a location on the screen specified by the user. This required students to use simple input statements to read the location, and provided ample opportunity to practice using a variety of colors and graphical primitives. Since the user had to have the ability to control the position of the face on the screen, students had to generate QuickDraw commands to draw the face using print statements that included both variables and string literals.

Students showed a significant level of creativity while completing this assignment. Some chose to reproduce well known characters from the mainstream media while others created innovative custom creations. In almost all cases, students expended significantly more effort than was required to meet the minimum specifications for the assignment which simply required the use of 4 colors and 4 different graphics primitives. This assignment was highly motivational, and gave students far more practice with the language syntax than we have achieved with any previous assignment that we have tried. We attribute this motivation to the creative and open ended aspect of the assignment, as well as the sense of competition that developed between some of the students in the class.

Subsequent assignments in the course were able to build on the students' previous experience working with QuickDraw. My colleagues and I have had students create and animate physics simulations such as a bouncing ball or a weight suspended from a spring. We have also had students graph mathematical functions or plot a grade distribution. These assignments have allowed us to evaluate students' ability to use control structures such as if statements and loops while solving problems that are more engaging than the problems that we have been able to use in an exclusively text based environment in the past.

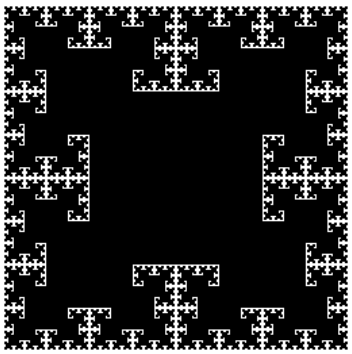

Figure 4. The Fractal T-Square

As we moved in to more advanced topics we continued to benefit from having the ability to use graphical assignments. For example, in our experience, students consistently struggle with both the concept of recursion and implementing a recursive algorithm that is provided to them. We believe that these struggles stem from not only the complexity of the topic, but also from poor engagement in the material due to the lack of compelling examples that can be presented during an introductory course. Because complex data structures haven't been introduced when recursion is first encountered, elegant examples of its use such as traversing a tree or implementing quick sort can't be presented.

Using QuickDraw has allowed us to consider interesting graphical problems that are well suited to being solved with recursion. Previous work has found that students are effectively motivated by problems such as finding a path through a maze [10] or drawing a fractal image [7, 8]. We have also had success with examples and assignments involving fractal images. For example, we have previously asked our students to complete an assignment where they generated the fractal T-Square shown in Figure 4. Our colleagues have also had success with asking students to generate the Hilbert curve, shown in Figure 5. In addition, we have found that students find implementing the flood fill or "paint bucket" tool to be an engaging graphical example of recursion which can be utilized once two dimensional arrays have been introduced.

When arrays and files are introduced we can consider additional interesting graphical problems. Previous studies have examined using image processing algorithms to reinforce these topics [2, 3, 13]. In these studies students have used Java, C++ and C. We have also found that introducing image processing operations such as gray scaling, ad-

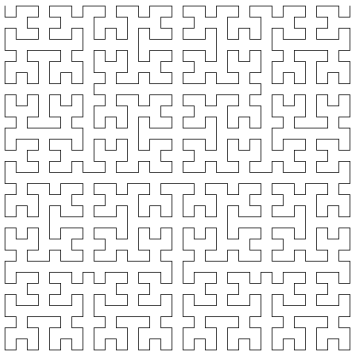

Figure 5. The Hilbert Curve

justing brightness, chroma keying, and rotation works well using Python and QuickDraw.

Implementing these operations forced students to work with nested loops and two dimensional lists. In addition, the operations that we selected formed a nice progression in complexity. When adjusting the brightness, each pixel in the output image is generated from the pixel at the corresponding location in the input image. Chroma keying builds on this idea by using pixels from two different input images. Rotation provides additional complexity by requiring students to use a pixel from a different location in the input image when computing the value of a pixel in the output image. For added challenge, students can be asked to complete transformations such as generating a pixelated image or zooming in on a portion of an image. For the final two transformations, one pixel in the input image is used to generate several pixels in the output image. When we used a selection of these problems as an assignment in CS1 we found that most students could complete the initial problem with modest effort while the final problems represented a significant, but solvable, challenge.

While we have highlighted some graphical problems that have worked particularly well as CS1 assignments for us, there are many additional visual problems that can be considered. These can be be used to reinforce topics such as object oriented design and more complex data structures in an engaging manner. When problems for these areas are discussed in the literature they are often presented using a specific library or tool. However the general ideas appear to be easily portable to Python and QuickDraw in many cases.

2.5 Strengths Summary

We have found that introducing Python and QuickDraw into CS1 has resulted in several advantages. As expected, we found that students find the graphical assignments more engaging than the text based assignments that we have used in the past. However, some unexpected benefits were also observed including increased readability of student code due to better indenting, and providing students with an easy to use environment for experimentation.

3. Drawbacks of Using Python and QuickDraw

While we have observed several significant positive effects by introducing Python and QuickDraw into CS1, we also encountered some minor, but unexpected, drawbacks. These drawbacks are described in the following sections, along with strategies for mitigating them.

3.1 Typos result in New Variables

As we noted previously, Python is a dynamically typed language which automatically creates a new variable when a value is assigned to a name that has not been used previously. While this feature allows students to begin programming without a detailed discussion of variable declaration and data types, it also makes it easy to introduce bugs that we have seen new programmers struggle to identify. In particular, a typo in a variable name on the left side of an assignment statement results in the creation of a new variable when the intention was to overwrite an existing value. Python's case sensitivity aggravates this problem, with the language viewing the variables `count` and `Count` as distinct. Unfortunately we are yet to find any way to mitigate this problem beyond emphasizing that programming is a detail-oriented activity, and that a single character error can be the difference between a completely non-functional program and a program that functions as desired. As students' programs become larger, the possibility of this kind of error occurring increases. However, students have also gained additional experience by this time, and as such, are more careful in their initial coding, and better at finding such bugs.

3.2 All Parameters are Passed by Reference

Python is a higher level language than those that we have used to teach first year computer science courses in the past. It doesn't expose pointers directly the way that C++ and Pascal do, making it difficult to discuss topics like dynamically allocating memory and dereferencing a pointer. In addition, all values in Python are objects, and all parameters to Python functions are passed as references to those objects. As such, it is it difficult to discuss the important distinction between pass by reference and pass by value semantics in a concrete manner. Further confusion is introduced because, while python does not support pass by value parameters, it does include immutable types which provide similar behavior for an entirely different reason.

When we used Python in CS1 for the first time we included a discussion of pass by value and pass by reference parameters. However, our impression was that even the best students got little out of this discussion because it wasn't possible to demonstrate the concept using Python. As a result, we have decided not to discuss pass by value and pass by reference parameters while teaching CS1 during the current term. This topic will be delayed until students encounter

a lower level language in CS2, which our department is presently teaching using C++.

3.3 Students Need to Learn Two Languages Simultaneously

QuickDraw commands consist of a command word followed by 0 or more parameters, some of which may be optional. Parameters are separated from the command word and each other using spaces. While this is conceptually similar to a Python function call, it is syntactically different in so much as Python encloses the parameter list to a function in round brackets and separates the parameters with commas. As a result, there are two distinct notations that students must become familiar with.

In most cases we have found that students do not struggle with the difference between these two languages. However, we have observed limited difficulties among a few weaker students. When such confusion has arisen, it has been easily cleared up with one or two concrete examples. As such, we view this as a minor concern that we may be able to completely avoid by including one or two additional examples in our early lectures.

3.4 Getting Python and QuickDraw Installed Requires More Skill

While our institution provides well equipped computer labs for our undergraduate students to use to complete their assignments, many students elect to work on their own computers. Just as using Python and QuickDraw requires students to learn two languages, setting up their own computer so that they can complete assignments requires them to install two separate pieces of software.

Python is easily setup under Windows using the installer from the Python website. However, the installer we have used does not automatically add Python's bin directory to the path. In our experience, this leads some students to believe that they have failed to install Python correctly because issuing the python command from the command prompt does not provide the behavior demonstrated in class. In reality, we have found that students often have Python installed correctly and simply need to make an addition to the path environment variable. While making this modification is reasonably easy, most of our students are completely unfamiliar with the concept of an environment variable, and as such, are unable to solve this problem without a clear set of instructions.

QuickDraw achieves its platform independence by being developed in Java. As a result, students need to have the Java runtime environment installed to use it successfully. While some students already have the Java runtime environment installed, many do not. Unlike Python, which provides a limited number of clear download options, Java's website at `http://java.sun.com` provides a myriad of download choices, making it difficult for students to determine which specific piece of software should be downloaded and installed. The confusion can be removed by directing students to `http://java.sun.com/getjava`. This website includes an automated tool which automatically determines which version of the Java runtime should be installed.

3.5 Teaching Graphics Concepts takes Time

Introducing graphical assignments has required students to become familiar with a number of graphics concepts that would normally be delayed until a later course. These include the concept of an image being formed from a collection of pixels, expressing colors as a combination of red, green and blue components, and the computer graphics coordinate system.

Of these concepts, we have seen students struggle the most with the orientation of the graphics coordinate system, we believe, because it behaves differently from the Cartesian coordinate system which students have worked with extensively in their prior mathematics courses. In particular, QuickDraw, like many graphics packages, sets the origin of the coordinate system in the upper left corner of the screen, with increasing values of y moving down the screen. While this is an initial source of confusion, we have found that students quickly adapt to the new system once it is explained to them. As such, the only drawback is the need to spend some time introducing these concepts.

3.6 Version Numbers are Important

Python is more fragile from version to version than the languages that we have worked with in the past. We were caught up on some differences between Python 2.4.3 and 2.5.1 such as the lack of the partition method on strings in the earlier version. Additional changes have been made for Python 3.0. While most of the changes are relatively minor syntactically, such changes in syntax result in programs that do not run, potentially causing much frustration for new programmers. As such, it is important to ensure that all lab machines, the instructor's machine, and any student personal computers are running the same version of Python. Students must also be strongly cautioned that any code that they find on the Internet may not run as desired because it was developed for a different version of Python.

3.7 Drawbacks Summary

This section has described several minor drawbacks that we encountered as a result of transitioning CS1 to Python and QuickDraw after teaching it with more traditional languages such as Pascal, Java and C++. Of these drawbacks, we found the most serious to be Python's automatic creation of new variables. Unfortunately, we do not presently have a good solution for this problem beyond reinforcing the need to exercise care while writing and debugging code. The remaining difficulties are all easily mitigated through appropriate examples, clear instructions, and a minor adjustment of the topics covered in CS1 and CS2.

4. Conclusion

While minor pitfalls were encountered as a result of our transition from traditional first year languages and assignments, we believe that the gains in student engagement and motivation which resulted from moving to Python and QuickDraw far outweigh the disadvantages. In particular, the opportunity to tackle larger assignments with a creative component increased students commitment to their work and resulted in some submissions that far exceeded the instructor's expectations. We have no doubt that students benefited from the extra time devoted to the assignments as it provided them with additional opportunities to practice both programming concepts and language syntax. Furthermore, many of the pitfalls that we have become aware of are easily avoided or mitigated when appropriate steps are taken.

References

[1] Sanjeev Arora and Bernard Chazelle. Is the thrill gone? *Commun. ACM*, 48(8):31–33, 2005.

[2] Kevin R. Burger. Teaching two-dimensional array concepts in Java with image processing examples. *SIGCSE Bull.*, 35(1):205–209, 2003.

[3] Jeffrey W. Chastine and Jon A. Preston. Teaching 2d arrays using real-time video filters. In *SIGITE '05: Proceedings of the 6th conference on Information technology education*, pages 133–137, New York, NY, USA, 2005. ACM.

[4] Stephen Cooper, Wanda Dann, and Randy Pausch. Alice: a 3-d tool for introductory programming concepts. In *CCSC '00: Proceedings of the fifth annual CCSC northeastern conference on The journal of computing in small colleges*, pages 107–116, , USA, 2000. Consortium for Computing Sciences in Colleges.

[5] Michael de Raadt, Richard Watson, and Mark Toleman. Language tug-of-war: industry demand and academic choice. In *ACE '03: Proceedings of the fifth Australasian conference on Computing education*, pages 137–142, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[6] Peter J. Denning and Andrew McGettrick. Recentering computer science. *Commun. ACM*, 48(11):15–19, 2005.

[7] Bruce S. Elenbogen and Martha R. O'Kennon. Teaching recursion using fractals in prolog. In *SIGCSE '88: Proceedings of the nineteenth SIGCSE technical symposium on Computer science education*, pages 263–266, New York, NY, USA, 1988. ACM.

[8] Aaron Gordon. Teaching recursion using recursively-generated geometric designs. *J. Comput. Small Coll.*, 22(1):124–130, 2006.

[9] R. Jiménez-Peris, S. Khuri, and M. Patino-Martínez. Adding breadth to CS1 and CS2 courses through visual and interactive programming projects. In *SIGCSE '99: The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 252–256, New York, NY, USA, 1999. ACM.

[10] Ivan B. Liss and Thomas C. McMillan. An amazing exercise in recursion for cs1 and cs2. In *SIGCSE '88: Proceedings of the nineteenth SIGCSE technical symposium on Computer science education*, pages 270–274, New York, NY, USA, 1988. ACM.

[11] Bill Manaris. Dropping cs enrollments: or the emperor's new clothes? *SIGCSE Bull.*, 39(4):6–10, 2007.

[12] Linda Mannila and Michael de Raadt. An objective comparison of languages for teaching introductory programming. In *Baltic Sea '06: Proceedings of the 6th Baltic Sea conference on Computing education research*, pages 32–37, New York, NY, USA, 2006. ACM.

[13] Sarah Matzko and Timothy A. Davis. Teaching CS1 with graphics and c. *SIGCSE Bull.*, 38(3):168–172, 2006.

[14] David A. Patterson. Restoring the popularity of computer science. *Commun. ACM*, 48(9):25–28, 2005.

[15] Lutz Prechelt. An empirical comparison of seven programming languages. *Computer*, 33(10):23–29, 2000.

[16] Rick Rashid. Image crisis: Inspiring a new generation of computer scientists. *Commun. ACM*, 51(7):33–34, 2008.

[17] Eric Roberts and Antoine Picard. Designing a Java graphics library for CS 1. *SIGCSE Bull.*, 30(3):213–218, 1998.

[18] Gerald Shultz. Integrating 3d graphics into early CS courses. *J. Comput. Small Coll.*, 21(3):169–178, 2006.

[19] Ben Stephenson and Craig Taube-Schock. QuickDraw: Bringing Graphics Into First Year. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 211–215, New York, NY, USA, 2009. ACM.

[20] Kelvin Sung, Michael Panitz, Scott Wallace, Ruth Anderson, and John Nordlinger. Game-themed programming assignments: the faculty perspective. *SIGCSE Bull.*, 40(1):300–304, 2008.