

PhrogLAB: Taking a Code-Centric Approach to Introductory CS Education

ABSTRACT

There is great concern in the technology industry and in university education about the dwindling pipeline of students entering information technology-related majors. One approach to stimulate interest is to create introductory courses that use “model-centric” programming, where the complexity of code development is mostly hidden. In spite of the success of tools such as Alice and Scratch, little beneficial impact has been felt thus far in the enrollment of students in computer science (CS) majors. One problem may be that students introduced to model-centric programming at the introductory level are less prepared to transition to professional-grade, “code-centric” languages. Taking instead a code-centric approach at the introductory level may encourage students to have a more realistic appreciation for the challenges of studying CS without being intimidated by writing code, and may provide the basis for further assessment of factors affecting efforts to increase in the number of students who pursue CS majors at the college level.

Categories and Subject Descriptors

D.3.3 [Programming Languages]: Language Constructs and Features.

K.3.2 [Computer and Information Science Education]: Computer Science Education.

General Terms

Measurement, Design, Human Factors, Languages.

Keywords

Introductory programming, novice programming environments, Phrogram, integrated development environment, games, storytelling.

1. INTRODUCTION

The shrinking pipeline of CS majors is an ongoing concern among employers and college educators, and there is significant under-representation in large segments of the population [1]. One of the most visible and ongoing efforts to reverse this trend has been the introduction of “model-centric” programming tools, emphasizing 3D graphics, animation, and storytelling; however, the ongoing decline in interest in the study of CS at the college level does not appear to have been significantly altered in recent years by the introduction of such tools.

One question that arises with respect to model-centric tools is how

well software development using them matches the coding and debugging practices that routinely engage professional programmers. An approach that introduces a “code-centric” view, while still emphasizing the creation of visually rich programs in both 2D and 3D environments, may have better results at keeping students interested in CS beyond introductory levels. The underlying theory of this approach is that by providing more direct and realistic exposure to the actual mechanics of CS at an early stage, students may develop a more enduring interest in the field. They will then be better prepared to learn more complex languages, and will not be discouraged by finding, or at least perceiving, a “disconnect” between model-centric programming activities and the syntax-oriented programming challenges they will inevitably encounter as they advance in their CS studies.

Phrogram is a programming language that introduces the concepts and practices of writing code in a robust, yet simplified integrated development environment (IDE), utilizing a “plain language” syntax with basic object-oriented capabilities, and taking a minimalistic approach to formal coding conventions. This paper introduces “PhrogLAB,” a teaching project using Phrogram at the high school level that is focused on learning how to write syntactically-correct code for both 2D and 3D programs, rather than manipulating 2D or 3D models in a “drag-&-drop” environment.

PhrogLAB is now under consideration for a grant award under the NSF’s BPC (Broadening Participation in Computing) initiative [2], which aims to expand participation in the information technology professions among members of under-represented communities. If funded through the BPC initiative, PhrogLAB will be piloted with students from Hispanic communities in West Texas and eastern New Mexico, and urban youth in working class neighborhoods of Seattle, WA, with a particular focus on attracting participation by girls as well as boys from these communities. The question of how well this type of approach may work with such students, or other students with limited exposure to programming, is deserving of further investigation.

2. MODEL- VS. CODE-CENTRIC

Over the past decade, educators have developed a myriad of tools to help novices learn programming, including narrative, visual and flow-model tools [3]. However, little focus appears to have been paid to the notion that pre-college level students can be effectively introduced to the coding aspects of programming, without impeding their interest in CS. While languages such as Alice and Scratch clearly have pedagogical value, they do not appear to have had any measurable effect on increasing the levels of college undergraduates who express a long-term interest in CS studies, notwithstanding the several years in which they have been used in secondary level and CS0 curricula. We question whether, by subordinating the writing of code as much as they do, languages that pursue a model-centric approach may in fact be hindering the transition of students to advanced college-level study, where they

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Conference’09, March 4–7, 2009, Nashville, TN, USA.
Copyright 2009 ACM 1-58113-000-0/00/0004...\$5.00.

must learn more complex CS concepts than are possible in mouse-driven, “drag and drop” environments.

The issue of which computer language to adopt for a first course in programming has been the subject of ongoing scrutiny [4]. Recently, attention has been focused on the value of concept visualization through virtual worlds, on the assumption that “environments rich in visual information facilitate learning better than text-based systems” [4]. The most prominent examples of this approach are Alice [5] and Scratch [6], which originated at Carnegie Mellon University and MIT respectively. They have found substantial adoption in introductory CS courses at both the college and pre-college levels. Alice in particular has attracted significant attention in recent years, because it is designed to offer a “first exposure to object-oriented programming” that allows students to create animated movies and simple games for 3D virtual worlds [5].

One concern expressed repeatedly by commentators evaluating teaching techniques for introductory CS courses is that the immediate exposure to, and emphasis on, writing code, invites difficulties because the use of incorrect data structures or the mis-application of language features “can turn an easy problem into a coding nightmare,” resulting in an undue perception that CS is intrinsically “hard” [7]. This assessment seems to echo a view held by one commentator at the dawn of CS education in the 1970s, that “the syntax of a programming language affects how it is used all out of proportion to its importance in the overall scheme of things” [8].

The pedagogical preference in recent years for the model-centric approach and the perceived challenges and deficiencies of introducing syntax to beginning programmers appear to be two sides of the same coin. That is, one rationale offered in support of teaching a programming language such as Alice is that a “drag-and-drop editor relieves the initial struggle with syntax that is often encountered by novice programmers” [9]. Conversely, the desire to eliminate syntax issues (“you don’t need to worry about where to put semi-colons and square brackets”) was an underlying design principle of Scratch [10]. However, as noted by one researcher, when transitioning from the model-centric environment of Alice to “the textual language and syntax” of Java, some students encountering difficulty came to believe “that their success in Alice had not been ‘real programming’ but rather just fooling with a toy environment designed for a younger audience” [11]. The issue here is whether the highly visual nature of Alice and Scratch, which may make them more enjoyable and less intimidating to beginning CS students, also gives them less confidence when they are thrust (as they must be, eventually) into the more complex world of syntax-driven, code-centric programming.

In our view, the challenge is to find something between an over-reliance on manipulating models in 2D or 3D environments, or “scenes,” and the code-centric reality of more powerful languages such as Java and C++ [12]. We believe that Phrogram offers such a middle ground, because it requires the programmer to pay careful attention to code, and provides the types of debugging tools (including “step into” and “step over” a current method) that are characteristic of a modern IDE, while also offering the immediate feedback of seeing the runtime output of the code through use of its built-in, “one-click” compiler.

3. GOAL: MAKE CODING FUN AS WELL AS CHALLENGING, WITHOUT BEING TOO COMPLEX

Much research has been done on what types of CS instruction are likely to have the most success in stimulating the current generation of high school and college-age students. Understanding that today’s younger generation has grown up with the Internet, video games, cell phones and an ever-increasing array of new technologies and gadgets, it is clear to most educators that the more primitive programming tools introduced a few decades ago cannot suffice to keep the interest of today’s CS students [13]. In contrast, the success of Alice in presenting a more compelling introduction to CS for at-risk students is noteworthy [14]. A key strength of a language such as Alice or Scratch noted by one commentator is its “tinkerability” – encouraging students to “mess with the materials” through iterative shifts in direction in the middle of a process [15]. We agree that encouraging open-ended experimentation is of critical importance for any programming language to be effective as a teaching tool.

Of equal, if not greater, importance is the ease with which subjects covered in CS education can inspire students. In this regard, the literature is now extensive on the value of using the creation of stories and games in CS education [12, 13, 16, 19]. PhrogLAB seeks to capitalize on this pathway by providing students with a sense of their potential for CS through gaining mastery of at least the basics of how to code a 2D or 3D game or story.

Some recent college-level courses have utilized the C# programming language and Microsoft’s XNA framework to develop game-themed programming assignments [17]. One obvious concern with this approach for a course geared to high school students (or any students from under-represented populations who are less likely to be familiar with modern programming) is that C# directly exposes the vast number of namespaces and classes offered by .NET, and thus “can be overwhelming for students with only a minimal exposure to programming” [18]. Given that presenting the challenges of writing code in a simplified and accessible way is essential to our approach, we do not consider a professional-grade language an ideal framework. But the relative merits of other options must also be considered.

4. EXPLORING THE OPTIONS

To teach high school students how to write code on a “gentle learning curve” and in a non-intimidating way, we considered several languages, apart from Phrogram, that might meet our pedagogical and research goals. The options focused primarily on what we would characterize as “scene-driven” (Alice and Scratch) and “game-driven” (DarkBASIC, Game Maker and Flash) design approaches. While there are also simplified Java-based learning languages (BlueJ and Greenfoot) that are both object-oriented and code-centric in nature, we did not explore them further because they do not natively support 3D programming [19], which we considered important for attracting high school students who are interested in programming for video game consoles.

4.1 Why not a Scene Design Approach?

In Alice, users “drag-and-drop words representing commands that objects in the 3D scene understand” [20]. Alice users manipulate

objects consisting of low-polygon models from built-in libraries whose high fidelity comes from hand-painted texture maps, so that adding external resources is not within the competency of most users [21]. In contrast, Phrogram uses common format sounds (.wav), images (.jpg) and models (.x) from external sources, so students are not limited to the built-in media assets that populate the virtual world they choose to initialize. The media assets are then manipulated through methods and calls that must be based in the code, not in the visual world environment. Phrogram also has a drawing library that gives users the ability to use lines, rectangles, circles and polygons in 2D game development. We believe this approach may better highlight the need to manage media assets in a programming project and the user's ability to directly control such assets through code has the potential to instill a deeper understanding of how modern programming is practiced as a professional discipline.

Scratch, which lets users "create programs by dragging and dropping individual action blocks that then snap together into scripts," is based on the concept of placing sprites on a "stage" and controlling their interactions in a scripting area [22]. Scratch has been used effectively to engage middle school students participating in brief programming classes [22] and also appears to have value as an introductory language to Java [6]. But there are some drawbacks in using it at the high school level that are similar to, and possibly more pronounced than, Alice. Like BlueJ and Greenfoot, Scratch is limited to 2D graphic formats (JPG, BMP, PNG, and GIF) and does not support several core features in modern programming, such as string concatenation and arrays; it also uses loop structures that are not based on familiar C-style terminologies such as "while," and "for" [23]. For these reasons, we do not view Scratch as the basis for a code-centric approach that will help students learn how to think syntactically in a programming language, as a foundation for the more advanced study of CS.

4.2 Why not a Game Design Approach?

There are several languages and tools designed to be useful in the creation of games. We considered three of the most prominent, but did not find them to have the right mix of code development and IDE exposure that we consider optimal for the code-centric learning approach we envision taking for PhrogLAB.

DarkBASIC, one of several simplified virtual-reality programming tools that have been used in educational settings, is designed to be coupled with 3D engines that provide a software library of 3D graphics functions. While tools in this category are useful for creating applications without the complexity of C++, which is widely used for professional-quality game programming, they are still designed for authors who already have development skills and their use is generally recognized as being "too complex for the average user without an extensive programming background" [24].

Game Maker, which has been used in high school and university CS0 classes, has the stated advantage of allowing students to build games and learn object-oriented design even if they "can't code a single line" [25]. GML, Game Maker's built-in scripting language, supplements its basic drag-and-drop system with deeper functionality. However, Game Maker is not intended to teach students how to "program" games, but rather how to create them and has limited 3D graphic support. It is also limited in its ability

to facilitate collaborative work, as the process of merging the work of two developers is "awkward, forcing all of the second developers' work into a sub-hierarchy of objects" [26]. Considering the value of having high school student programming projects be team-based, and the lack of a coding component in Game Maker's value proposition, it is clearly not a good fit for the kind of code-centric approach we wish to take.

Adobe Flash, a multimedia authoring tool used to develop animation and interactivity for the web, has been widely used for game development and offers interesting advantages in teaching programming at the high school level, most notably its integration of a powerful scripting language (ActionScript), being web-based and having an extensive list of learning resources and tutorials [27]. However, Flash is not code-based and like C#, ActionScript is not designed for novice programmers; when studied together, the overall robustness and complexity of Flash and ActionScript lend themselves more toward a CS1 to CS2 level course, as opposed to high school or the CS0 level [28].

For these reasons, we did not find any mainstream tools focused on scene or game design that fit our desired approach, which is to present a "code-centric" view of programming at the high school level so that students from under-represented populations may be exposed to syntax-related challenges in a way that helps them appreciate and grasp them better when their CS studies progress to more powerful, professional grade languages such as C++ and Java.

4.3 Why Phrogram?

Phrogram's language and IDE can present the core patterns and practices of modern software development to students at the high school and CS0 level, thereby giving them a stronger initial footing as they gradually enhance their programming skills. While being similar in some respects to other languages that introduce CS concepts through games and stories, Phrogram differs fundamentally from languages that rely on a model-centric, "drag & drop" approach, and other environments that are not focused on the actual process of writing code at the novice level.

Phrogram covers the essential code-creation nature of programming in the context of games and stories, without the complexity of a professional-grade IDE, and without the limitations of a product optimized solely for the purpose of creating – but not programming – games. If students begin to understand how they can realistically aspire to become programmers by learning how to write code for games or other types of graphically rich programs, they may begin to appreciate that learning CS is not as difficult and mysterious as they might otherwise think.

Phrogram's code-centric environment is comparable to the source (as opposed to design) view found in tools most widely used by working programmers, such as Microsoft Visual Studio and Eclipse. By emulating what modern programming practices are all about, and by focusing on problem-solving as it relates to a natural-language syntax, PhrogLAB may be able to convey a more realistic understanding of what high school students with limited exposure to programming can expect if they decide to pursue the study of CS at higher levels.

Phrogram's current 2.x version encompasses programming using both 2D or 3D graphics. A 3.0 version, now in development and slated for release in late 2008 or early 2009, will upgrade its 3D

libraries so that they are based on Microsoft XNA, thereby giving users the ability to create games for Xbox 360 which can be offered (through paid membership in Microsoft's "Content Creator Club" program) in Microsoft's Xbox Live Marketplace.

Phrogram is currently being taught at schools from the elementary to the college level, including Ohio State University and the University of Michigan. Over the last 12 months, it has been used for a class offered by Red Llama, Inc in a community center in a low-income Seattle neighborhood. In Red Llama's program, middle school students in Seattle's Yesler and Delridge neighborhoods came twice per week, over a 5-week period, to learn how to create PC games with 2D graphics. While not rigorous, informal evaluation suggests that this program had a positive impact on perceptions of CS by participants. Pre- and post-completion surveys asked students on a scale of 1 to 5 to consider the truth of the statement, "I want to be a technology professional," with the average score rising from 3.7 to 3.9. In rating the truth of the statement, "I feel confident in my technical skills," the score rose from 3.6 to 3.9. Qualitative data collected at the end of the class in response to the question, "Was this program beneficial for you? How?" – yielded 100% positive responses. An example of a student quote is: Yes – it just helps me bring stuff outside of myself – like ideas/interests...like, I've never thought I'd design a game, but I got into it..I'd like to create more games" [29].

5. CONCLUSIONS

PhrogLAB aims to teach modern programming concepts using a language that, while not as powerful as C++ or Java, supports object-oriented methodologies, offers the experience of IDE-based development, and allows for the creation of stand-alone executable programs (capable of being run outside Phrogram's IDE). PhrogLAB will present the core challenges of programming - coding, error detection, debugging, object manipulation, and maximizing performance efficiencies – at the high school level in a code-centric way. Students will assess their work on the basis of an end-to-end understanding of how to code, and how to address errors that arise in their work either at the coding or compiling stage. While Phrogram is a commercial product, licensing costs for its use in PhrogLAB are expected to be a relatively minor component of the overall budget. The use of Phrogram allows assessment of a paradigmatic educational approach that differs fundamentally from the heavily-explored approaches embodied by visually-oriented, drag-and-drop style tools such as Alice and Scratch, while still focusing on code creation at the novice programmer level.

6. ACKNOWLEDGEMENTS

We would like to thank Andy Dunn of The Phrogram Company for his assistance in helping to formulate the core ideas behind our BPC proposal, and in offering us guidance on the capabilities of, and the thinking behind, Phrogram.

7. REFERENCES

- [1] Grose, T. (2006, October). Trouble on the Horizon. ASEE Prism, 16(2), http://www.prism-magazine.org/oct06/feature_trouble.cfm
- [2] <http://www.nsf.gov/pubs/2007/nsf07548/nsf07548.htm>
- [3] Powers, K., Gross, P., Cooper, S., McNally, M., Goldman, K. J., Proulx, V., and Carlisle, M. 2006. Tools for teaching introductory programming: what works?. SIGCSE Bull. 38, 1 (Mar. 2006), 560-561. DOI= <http://doi.acm.org/10.1145/1124706.1121514>
- [4] Duke, R., Salzman, E., Burmeister, J., Poon, J., and Murray, L. 2000. Teaching programming to beginners - choosing the language is just the first step. In Proceedings of the Australasian Conference on Computing Education (Melbourne, Australia). A. E. Ellis, Ed. ACSE '00, vol. 8. ACM, New York, NY, 79-86. DOI= <http://doi.acm.org/10.1145/359369.359381>; Kelleher, C. and Pausch, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv. 37, 2 (Jun. 2005), 83-137. DOI= <http://doi.acm.org/10.1145/1089733.1089734>
- [5] Dougherty, J. P. 2007. Concept visualization in CS0 using ALICE. J. Comput. Small Coll. 22, 3 (Jan. 2007), 145-152
- [6] Malan, D. J. and Leitner, H. H. 2007. Scratch for budding computer scientists. SIGCSE Bull. 39, 1 (Mar. 2007), 223-227. DOI= <http://doi.acm.org/10.1145/1227504.1227388>
- [7] Rao, T.M. and Mitra, S. 2008. An Early Software Engineering Approach to Teaching CS1, CS2 and AI. In Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (Portland, OR, USA, March 12 - 15, 2008). SIGCSE '08. ACM, New York, NY, 143-147. DOI= <http://doi.acm.org/10.1145/1352135.1352185>
- [8] Plauger, P. J. 1975. Signal and noise in programming language. In Proceedings of the 1975 Annual Conference ACM 75. ACM, New York, NY, 216
- [9] Cooper, S., Dann, W., Pausch, R. 2003. Using animated 3D graphics to prepare novices for CS1. Computer Science Education, 13(1). http://www.sju.edu/~scooper/alice/scooper_csej.pdf
- [10] Resnick, M. 2007. All I really need to know (about creative thinking) I learned (by studying how children learn) in kindergarten. In Proceedings of the 6th ACM SIGCHI Conference on Creativity & Cognition (Washington, DC, USA, June 13 - 15, 2007). C&C '07. ACM, New York, NY, 1-6. DOI= <http://doi.acm.org/10.1145/1254960.1254961>
- [11] Powers, K., Ecott, S., and Hirshfield, L. M. 2007. Through the looking glass: teaching CS0 with Alice. SIGCSE Bull. 39, 1 (Mar. 2007), 213-217. DOI= <http://doi.acm.org/10.1145/1227504.1227386>; see also: Redden, E. May 22, 2007. Programming with pictures, <http://www.insidehighered.com/news/2007/05/22/compsc>
- [12] Gonzalez Sanchez, J., Berrelleza Perez, R. A., and Chavez Echeagaray, M. E. 2007. Introducing computer science with Project Hoshimi. In Companion To the 22nd ACM SIGPLAN Conference on Object Oriented Programming Systems and Applications Companion (Montreal, Quebec, Canada, October 21 - 25, 2007). OOPSLA '07. ACM, New York, NY, 908-914. DOI= <http://doi.acm.org/10.1145/1297846.1297942>
- [13] Anderson, E. F. and McLoughlin, L. 2007. Critters in the classroom: a 3D computer-game-like tool for teaching programming to computer animation students. In ACM SIGGRAPH 2007 Educators Program (San Diego, California,

- August 05 - 09, 2007). SIGGRAPH '07. ACM, New York, NY, 7. DOI= <http://doi.acm.org/10.1145/1282040.1282048>
- [14] Moskal, B., Lurie, D., and Cooper, S. 2004. Evaluating the effectiveness of a new instructional approach. SIGCSE Bull. 36, 1 (Mar. 2004), 75-79. DOI= <http://doi.acm.org/10.1145/1028174.971328>
- [15] Resnick, M. and Silverman, B. 2005. Some reflections on designing construction kits for kids. In Proceedings of the 2005 Conference on Interaction Design and Children (Boulder, Colorado, June 08 - 10, 2005). IDC '05. ACM, New York, NY, 117-122. DOI= <http://doi.acm.org/10.1145/1109540.1109556>
- [16] Cliburn, D. C. and Miller, S. 2008. Games, stories, or something more traditional: the types of assignments college students prefer. In Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (Portland, OR, USA, March 12 - 15, 2008). SIGCSE '08. ACM, New York, NY, 138-142. DOI= <http://doi.acm.org/10.1145/1352135.1352184>; Kelleher, C., Pausch, R., and Kiesler, S. 2007. Storytelling Alice motivates middle school girls to learn computer programming. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (San Jose, California, USA, April 28 - May 03, 2007). CHI '07. ACM, New York, NY, 1455-1464. DOI= <http://doi.acm.org/10.1145/1240624.1240844>
- [17] Sung, K., Panitz, M., Wallace, S., Anderson, R., and Nordlinger, J. 2008. Game-themed programming assignments: the faculty perspective. SIGCSE Bull. 40, 1 (Feb. 2008), 300-304. DOI= <http://doi.acm.org/10.1145/1352322.1352241>
- [18] Chaytor, L. and Leung, S. 2003. How to creatively communicate Microsoft.NET technologies in the IT curriculum. In Proceedings of the 4th Conference on Information Technology Curriculum (Lafayette, Indiana, USA, October 16 - 18, 2003). CITC4 '03. ACM, New York, NY, 168-173. DOI= <http://doi.acm.org/10.1145/947121.947160>
- [19] Baldwin, R. Getting Started with the Greenfoot Java IDE, <http://www.developer.com/java/other/article.php/3761811>; Dann, W., Dragon, T., Cooper, S., Dietzler, K., Ryan, K., and Pausch, R. 2003. Objects: visualization of behavior and state. SIGCSE Bull. 35, 3 (Sep. 2003), 84-88. DOI= <http://doi.acm.org/10.1145/961290.961537>
- [20] <http://www.alice.org/stage3/projects.html>
- [21] Conway, M., Audia, S., Burnette, T., Cosgrove, D., and Christiansen, K. 2000. Alice: lessons learned from building a 3D system for novices. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (The Hague, The Netherlands, April 01 - 06, 2000). CHI '00. ACM, New York, NY, 486-493. DOI= <http://doi.acm.org/10.1145/332040.332481>; Sebe, I. O., You, S., & Neumann, U. (2005). Rapid part-based 3D modeling. Proceedings of the ACM Symposium on Virtual Reality Software and Technology (143-146). New York: Association for Computing Machinery. DOI= <http://doi.acm.org/10.1145/1101616.1101646>
- [22] Sivilotti, P. A. and Laugel, S. A. 2008. Scratching the surface of advanced topics in software engineering: a workshop module for middle school students. In Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (Portland, OR, USA, March 12 - 15, 2008). SIGCSE '08. ACM, New York, NY, 291-295. DOI= <http://doi.acm.org/10.1145/1352135.1352235>
- [23] <http://scratch.mit.edu/forums/viewtopic.php?id=313> (lack of string concatenation functionality in Scratch); http://www.ce.ucsc.edu/~karplus/scratch_programs/ (lack of support for arrays in Scratch); <http://www.developer.com/java/other/article.php/3761811> (unconventional loop structures in Scratch)
- [24] Moreno, G., Serra, J. L., et al. (2007). A documental approach to adventure game development. Science of Computer Programming, 3-31. Retrieved May 15, 2008, from http://www.e-ucm.es/drafts/e-UCM_draft_8.pdf
- [25] Overmars, M. (2004). Teaching computer science through game design. Computer, 37(4), 81-83.
- [26] Whitehead, J. 2008. Introduction to game design in the large classroom. Proceedings of Third International Conference on Game Development in Computer Science Education, Miami, FL (Feb 27-March 3, 2008). ACM, New York, NY, 61-65.
- [27] Sands, M., Moukhine, N., and Blank, G. 2008. Widening the pipeline of K-12 students with flash. J. Comput. Small Coll. 23, 5 (May. 2008), 52-57.
- [28] Leutenegger, S. T. 2006. A CS1 to CS2 bridge class using 2D game programming. J. Comput. Small Coll. 21, 5 (May. 2006), 76-83.
- [29] Gruenbaum, P. (2008). Reaching Low-Income Teens through Computer Programming, poster presented at SIGCSE Technical Symposium on Computer Science Education, Portland, OR (March 2008).