Read, Write, Play: An Educational Mobile Gaming Platform*

Jennifer Bayzick, Bradley Askins, Sharon Kalafut, and Michael Spear
Computer Science and Engineering Department
Lehigh University
{jlb411, bja206, smk5, spear}@cse.lehigh.edu

ABSTRACT

We introduce ALE, a new framework for writing games for the Android platform. The primary motivation behind ALE is to emphasize reading code before writing it. Beginners read game code to learn how levels can be made, and advanced users read the code of ALE itself to learn how to create useful and extensible libraries. To date, roughly 200 students at our university have used ALE, ranging from first-semester engineering undergraduates through Masters students. ALE has proven useful in teaching non-majors about CS, in making introductory CS programming courses more exciting, and in encouraging creativity, entrepreneurship, and good program design in upper-level electives. Based on these experiences, we encourage educators at all levels to consider using ALE to improve students' ability to learn by reading code.

Categories and Subject Descriptors

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

General Terms

Design, Languages

Keywords

Android, CS1, Games, Java, OpenGL

1. INTRODUCTION

While the authors of this paper are generally optimists, we worry that too many of today's CS students are not passionate about programming. They see it as drudgery. They do not enjoy it. They might abandon CS to avoid it. If not, too often they remain in CS but never learn to program well. And while programming is only a part of CS, it is an integral part. Just as calculus cannot be hidden from math majors forever, neither can programming be hidden from CS majors. In this paper, we describe our effort to make learning to program more fun.

We argue that there are four causes of this problem that are understudied, all of which can be addressed in part through a new framework we developed for Android game development, called ALE (the AndEngine Lehigh Extension). To date, we have employed projects based on ALE at three points in the curriculum: an 18-hour first-year experience course for first-year engineers, an introductory programming course for CS majors, and an upper-level mobile programming elective. Though informal, our experiences have been positive enough that we are now releasing ALE to educators worldwide, so others may benefit from our effort.

Projects involving ALE are fun, social, and output-focused. By design, they address the following four concerns:

- Students do not read source code, and indeed do not consider it a valuable resource for learning. Likewise, faculty do not treat source code as a pedagogical tool.
- Students do not write code to test hypotheses, only to produce solutions.
- Early curriculum programming projects fail to make programming a form of entertainment and self expression.
- Many instructors are not willing to redesign an entire course, but will integrate new ideas that are packaged as standalone projects requiring little infrastructure or support.

ALE is a Java-based platform for developing 2D games for the Android platform. It is an intermediate library that sits between the programmer and the complex AndEngine OpenGL ES 2.0 game development framework [8]. Using ALE, beginners can produce a variety of styles of game, with each level requiring as little as a dozen lines of declarative code. In this project configuration, students read the code of a sample game, select those levels that possess the features they desire for their game, and then write code using on a simple copy-paste-modify model. Unique and enjoyable games can be created without writing a single loop or defining a single class: only basic uses of variables and methods are required.

Advanced courses can require students to extend ALE, either by modifying and combining existing ALE features, or by providing clean new abstractions built directly atop AndEngine. Again, the emphasis is on reading code first, and writing code second. However, at this level arbitrarily advanced programming topics can be taught, to include events, callbacks, and asynchrony.

ALE encourages social interaction and self expression. Students must find or create their own artwork, music, and sound effects. Students work together to design their games, and then they debug their code by playing games together. When logic bugs manifest, they usually have an amusing visual manifestation (e.g., throwing the puppy to the stick, instead of throwing the stick to the puppy).

Furthermore, ALE is a pure Java project. It uses familiar, indus-

*This work was supported in part by the US National Science Foundation under grant CNS-1016828; and by financial and equipment support from Google.

SIGCSE 2012 Denver, Colorado USA

try standard tools that we bundle in a pre-packaged virtual machine. Since it is fun, students learn to use ALE quickly. Consequently, it is possible to use it as a capstone or enrichment experience, without redesigning an entire course around it. This also allows ALE to be integrated at many points in the curriculum without risking overexposure or watering down the curriculum.

In this paper, we discuss our motivation in designing ALE. We then provide some details about the framework, followed by a discussion how ALE can be integrated into curricula. We conclude with lessons learned and suggestions for instructors looking to increase excitement and improve programming proficiency without redesigning entire courses or learning complex software packages.

2. MOTIVATION AND RELATED WORK

Over the last few years, both mobile programming and game design have been embraced by educators seeking to increase excitement about CS and increase its appeal to a diverse audience. Much of this work forms the motivation for ALE, though there are some noteworthy distinctions. Below we discuss the industrial, social, and administrative trends that shaped the design of our project.

2.1 Industrial Trends

Nothing is from Scratch: In industry, a programmer usually starts with existing code, and then bends and changes it to create something new. Yet in many academic game projects, students still write the core game loop and low-level simulation code. desJardins and Littman note that too much programming decreases the appeal of a project to non-majors [4]. Particularly in projects that entice students to study CS, every line of code should directly correlate to an observable output.

Collaboration is the Norm: Not only do modern programmers typically work with other programmers, they also often must work closely with experts in other disciplines [28]. If students are not permitted to use copyrighted material, then there is an incentive to either develop friendships with students majoring in digital design and music, or to explore their own latent artistic abilities, even if only to modify royalty-free media. We believe that the interaction between art and code, born out of the interaction between artist and programmer, is the source of value in many programs, particularly those with which students are most familiar.

Code Writers Must Read Code: In order to extend existing software, a programmer must learn about its behavior. This typically comes from three areas: reading code itself, reading on-line documentation, and reading generated documentation (e.g., JavaDoc) from within an IDE. Professional programmers are expected to generate both code and documentation, and to review their peers' code. Preparing for these roles requires a fluency in reading code that is rarely emphasized.

Programming is not the Only Job Skill: Today's students worry about employment upon completion of their degrees. While CS jobs are plentiful, employers have high expectations. Projects that use industry-standard languages and tools, and that result in marketable software, distinguish job applicants. As a simple example, a student who has independently released software on a mobile application marketplace offers an employer evidence of her ability (measured in the program's ranking and number of downloads), creativity, customer focus, and entrepreneurship.

2.2 Social Factors

In addition, the following social factors play a significant role in student outcomes. To be clear, we believe today's student possesses tremendous ability and skill, and the discussion below is not intended to imply a negative attitude towards them. In raising these issues, we are instead sharing an opinion about student motivation and attitudes that we have formed through interactions with hundreds of students, both CS majors and non-majors. These observations are, at least at our university, an accepted fact. Our project treats them as an opportunity.

Fun Projects Matter: Throughout the curriculum, the use of mobile phones in general, and Android in particular, leads to an increased perception that projects are fun and relevant to future employment prospects. Andrus and Nieh observed this in an upper-division operating systems class [2], and Loveland similarly noted that both mobile programs and the Google brand made HCI projects more "cool" [13]. Independently, mobile programming [6, 7, 18, 27], graphics [9], and computer games [19–21,26] have been shown to improve student experiences in introductory courses. Clearly, mobile games are the sweet spot for maximizing enthusiasm, especially since they are *easier* to write than traditional games [11].

Students Expect Instant Gratification: Wolber discusses how the hour-long effort of getting a first "hello world" program working from scratch is anticlimactic, and might repel prospective students [27]. Today's students are too fluent in media-rich computing for such an introductory experience to succeed. If the effort required to create early programs seems too great, and the resulting software artifacts appear too simple, students may develop a sense of hopelessness and feel that creating production-quality software requires a skill set that is beyond their ability.

Passion Matters: Assignments with a single "right answer" stifle creativity. In such assignments, students are reluctant to push themselves toward abstract goals such as quality, user experience, and robustness. While these goals are difficult to grade, failing to encourage their pursuit is dangerous. Without them, students will not develop the enthusiasm observed by Greenberg et al., whose students would give themselves extra assignments, due to the passion they developed through their projects [9].

Bragging is OK, if not Good: Finally, programming projects should appeal to basic human nature. Students gain a sense of satisfaction, and departments receive free publicity, when projects result in tangible artifacts that students can show to peers. This has certainly been a common trend in engineering courses, yet CS courses rarely result in "boast-worthy" projects or useful take-home artifacts.

2.3 Administrative Constraints

Finally, many administrative constraints come into play. These issues deal with instructors and institutions, and capture the sources of reluctance that prevent even exceptional educators from adopting innovative material.

System Administration is not Free: Course administration has a real cost, particularly in sysadmin time. Projects that require custom software, or a particular operating platform, are difficult to adopt [5, 22]. The problem is worst when adding one project increases the support burden, as is the case when adding innovative courseware incrementally.

Service Expectations do not Include Games: Not all deans are as open-minded as ours, and as willing to allow 25% of incoming engineers to participate in a project involving games. Other

departments, similarly, may worry that too much emphasis on art, music, and game-play will not satisfy learning objectives related to computational science and engineering, particularly in departments with a substantial "service" enrollment.

Pedagogy Cannot be Sacrificed: New projects must still support core educational objectives. Malan discusses the redesign of CS1 at Harvard, and notes the danger of reducing the pedagogical value of courses in order to broaden participation [14]. A project must not require instructors to devote lecture time to framework details, or encourage them to de-emphasize learning objectives that do not lend themselves to projects within the framework.

Expertise is Limited: Finally, Sung [22] argues that it is difficult for faculty without game or graphic experience to get involved in mobile game projects. This suggests both that gradual adoption is a necessary strategy, and that the underlying framework cannot be too complex. The same social factors that affect our students apply here: faculty want fun projects, instant gratification, and the ability to boast to their peers. If a project is not easy for an instructor to master, it will not be useful in her classroom.

3. ALE DESIGN

3.1 System Administration

ALE's requirements are simple: inspired by Laadan et al. [12], we provide a Linux virtual machine (VM) with Eclipse, the Android SDK, a web browser, and image and audio editors. Each VM also includes a checkout of the ALE repository. The VM infrastructure eliminates the need for phone-specific drivers, or indeed any software on the host apart from Oracle VirtualBox (VMWare could easily be substituted). Installing ALE consists of (a) installing VirtualBox, (b) copying a single file (the VM image), and (c) running a 10-line non-interactive script to import and configure the VM. Since we use Android, there is no lock-in [5]: the host can use Windows, Linux, or MacOS.

3.2 Core Objects

ALE exposes 11 core objects, with which games can be written in a declarative fashion. These objects are grouped as follows:

First, the `Level`, `Media`, and `Controls` objects are used to define the shape of a level, the forces that occur on objects in that level, the images and sounds used by that level, and the level's on-screen controls (buttons and timers). In all, there are 36 methods for these purposes, though a typical game will use less than 10.

Second, the `Hero`, `Goodie`, `Enemy`, `Obstacle`, `Destination`, and `Bullet` objects are used to place individual entities into a game level. These entities indicate the main character in the game, items that must be collected, characters that must be avoided, barriers that prevent passage across the screen, a goal that the hero must reach, and particles that the hero can shoot at enemies. The interactions among these objects are complex, but are configured only through declarative code. All aspects of interaction are handled invisibly by AndEngine's physics simulator. 75 methods are available for placing and configuring these entities.

Third, the `HelpScene` and `PopUpScene` objects are used to generate on-screen help and tips. These features are simpler than those for defining levels, and we do not discuss them further in this paper. Their use entails up to 7 methods.

3.3 Game Configuration

Students modify a (well-commented) XML configuration file to specify game configuration. The file exposes 12 properties, relating to the customization of strings (such as the game name), the number of levels, and the orientation of the game (portrait or landscape).

```
public class MyGame extends ALEGame {
  public void nameResources()    {
    // see figure 3
  }
  public void configureLevel(int whichLevel) {
    // use Level, Controls, and 6 entity
    // object types to define each level
    if (whichLevel == 1) { // see figure 2 }
    else if (whichLevel == 2) { ... }
    ...
  }
  public void configureHelp(int whichScene) {
    // use HelpScene object to draw
    // help screens
    if (whichScene == 1) { ... }
    else if (whichScene == 2) { ... }
    ...
  }
}
```

Figure 1: ALE game shell. There are only 3 methods, which load images, draw help screens, and describe levels.

```
// Set screen size, max/initial gravity, & tilt
Level.reset(460, 320, 10, 10, 0, 0, true);
// Put a border on the screen. (1, .3f, 1) are
// density, elasticity, and friction
Obstacle.drawBorder(0, 0, 460, 320, "black.png",
    1, .3f, 1);
// Hero starts at (40,70), is 30x30 in size
Hero.addHero(40, 70, 30, 30, "hero.png",
    1, 0, 0.6f);
// Put the destination in the top left
Destination.addDestination(290, 60, 10, 10,
    "bullseye.png", 1, 0);
// Draw a wall that isn't bouncy
Obstacle.addStationarySquareObstacle(50, 0, 10,
    200, "wall.png", 1, 0, 1);
// Win when 1 hero reaches the destination
Level.setVictoryDestination(1);
```

Figure 2: Java code for a level of a simple tilt-based maze game.

3.4 The ALE Game Shell

While ALE exposes a callback mechanism, we exclude it from this discussion. Not counting callbacks, ALE games consist of only three methods, as depicted in Figure 1. Each level is declared within the corresponding `if` block. For example, Figure 2 shows the code for a maze level, with the main character in the top left, the goal in the top right, and a vertical wall between them. Only 6 lines of code are required, with no variables or control flow.

The parameters for this code fall into three categories: (x,y) dimensions specifying the location of an object, (width,height) dimensions indicating the size of an object, and (density, elasticity, friction) values to govern the behavior of an object when it experiences a collision. Since the underlying methods carry JavaDoc comments, the Eclipse IDE generates tooltip-style documentation for these fields to aid the programmer. Extending the behavior of an object is achieved with syntax such as:

```
Destination d = addDestination(...);
d.applyRotation(...);
```

ALE consists of 2891 lines of code and 3036 lines of comments.

```
// sound when the hero reaches the destination
Media.registerSound("happy.ogg");
// background music: ''true'' means to loop
Media.registerMusic("music.ogg", true);
// load the images that are used in level 1
Media.registerImage("black.png");
Media.registerImage("hero.png");
Media.registerImage("bullseye.png");
Media.registerImage("wall.png");
```

Figure 3: Registration of media files for simple tilt-based maze.

Its primary purpose is to hide AndEngine behind a clean, simple, and well commented interface. ALE supports a wide array of game types through just 118 methods. Levels can be infinitely long, heroes can shoot, jump, and duck, there is a health scoring system, levels can have time limits, and a variety of game types are possible. To date, students have built side-scrollers (like Temple Run), platform games (pinball, mazes), puzzles that involve creating a path for a laser beam to travel (laserlight), and games involving shooting falling/moving enemies (asteroid).

4. CURRICULAR INTEGRATION

ALE can be used in CS projects at any level. Below, we discuss recent and ongoing uses of ALE at our University.

4.1 Undeclared First-Year "Projects" Courses

At our university, all engineering students arrive with an undeclared major, and in the course of a common first-year curriculum, participate in a "projects" course. Through this course, they gain exposure to engineering disciplines by participating in two 6-week projects. Each project meets for one 3-hour lab per week. No prior knowledge is assumed.

This project allows us to reach a very diverse audience. We use ALE as the basis for a project titled "Mobile Game Programming for Fun and (non)Profit". Students have 6 weeks to produce a game that *could be* sold to raise money and awareness for the cause of their choosing. By merging gaming with social relevance, we accrue benefits similar to those observed by Pauca and Guy [16], while accommodating the fact that "fun" is more valuable than "important" to students [17]. By letting students choose their theme, we are, to a degree, following the advice of McFall and DeJongh, who use students' personal interests to make CS attractive [15], though we differ in that programming remains the focal activity.

Our goals are to reinforce general problem solving skills and to demystify programming. Recognizing that our class population is primarily non-CS engineers, the course is oriented toward non-majors, employing practices recommended by Adams and Pruim [1].

The only rules we impose are that (a) no copyrighted material can be used, (b) the game must relate to a social cause, and (c) games must be violence-free and gender-neutral [3, 22].

Teams of three work to produce a game. They begin by playing a sample game, for which they have the source code. This game consists of 41 levels that demonstrate ALE features, each requiring between 6 and 33 lines of code (the average is 12.5). Students decide which levels possess features that they would like to integrate into their game. They then base their game on the source code for those levels. We also provide two on-line websites. The first is an 18-page guided walk-through of the behavior of the sample game. The second is a more traditional reference, consisting of 17 documents (32 pages total) organized by ALE object types.

Our experience matches past observations [10, 25], in that students tend not to need much lecture, but that guided programming coupled with hands-on interaction with the professor and lab assistant results in rapid progress. We encourage all students to program, critique programs, and design levels.

A few of the sample levels include variables, and two include `for` loops in order to simplify the drawing of geometric patterns. A limited number of levels also include frame-based animation, which requires the use of simple arrays. Otherwise, individual levels require only declarative, straight-line code. For the most part, programming only entails placing objects on the screen, and configuring the images and sounds that are used. However, since a game can have an arbitrary number of levels, with each level arbitrarily long and complex, we have no trouble keeping even advanced students occupied for the entire 6-week project.

To summarize, the main goals of the project at this level are:

- Make computing fun
- Foster teamwork and relationship-building
- Demystify programming for engineers
- Encourage students to consider a major or minor in CS
- Ensure that each team completes the project
- Emphasize general problem solving skills
- Reward attention to detail and creativity

These goals are achieved by having students *read* source code, and then modify it. Self-expression and enjoyment are encouraged, especially by debugging (i.e., playing) each others' games.

4.2 Introductory Programming Courses

In introductory programming courses, we use ALE in a similar fashion. A key difference is that ALE is reserved for the end of the semester, when students will be able to enjoy it the most. When two weeks remain in class, we assign the ALE project, with the hope of mimicking the "wax on, wax off" trope [23]: after enduring a semester learning to program using traditional techniques, students discover that they've actually learned enough to write games!

Since students have much more fluency in Java, we expect more of them. We require them to use arrays for animation, and to use `for` loops to draw patterns of obstacles. We offer extra credit for using the callback mechanism, which may, in turn, require students to declare new fields and use references with complex object lifecycles. As before, we do not require modification of ALE itself, and instruction is primarily through on-line materials, not lectures. Students are encouraged to refactor the single `configureLevel` method through additional functions that improve both code reuse and organization. They also see first-hand the value of code documentation, since ALE's comments, which they encounter as tooltips in Eclipse, directly improve their productivity.

Grading involves two parts. First, the instructor grades the code according to the standards of the class. Second, students share their games with the class, which rates games subjectively, based on perceived quality and a "fun factor". Students learn about quality, since the most complex source code does not necessarily produce the highest-rated games.

Using ALE in this manner does not water down the course. All of the rigor of the original course is preserved, but ALE makes the final project more fun. This avoids the pitfall raised by Malan [14]: without sacrificing quality, a final large (800+ lines of code) project can be a form of entertainment and self expression.

4.3 Upper-Level Courses

Whether or not they are familiar with ALE from previous experiences, students in upper-level courses can also benefit. In our mobile programming class ALE provides a gentle introduction to the mobile development environment: the framework hides the An-

droid SDK, but the games themselves expose the nuance required to use touch and small screens effectively. Thus ALE helps get students into the right mindset: they have to think about the relationship between the device and the experience of using the application, but can do so without having to simultaneously learn complex Android APIs. Emphasizing marketability helps show students the importance of an intuitive interface and visual appeal.

ALE also encourages students to think about software development in general, and the design of reusable libraries in particular, making it a convenient project for reinforcing self-reflection about development [24]. In a software engineering course, students can be tasked with expanding ALE with new features. These often entail reading hundreds of lines of code, and then realizing that 10 lines or less suffice to add a powerful new feature. In reading code and navigating the layout of the library, students gain exposure to the fruit of a good software engineering effort, which then provides a case study for their own work. Their work can also be contributed back to ALE as open-source. A more complex requirement is to extend ALE to provide a clean interface to more of the underlying AndEngine framework. In truth, this assignment is somewhat sadistic, in that ALE is well commented, whereas AndEngine is virtually comment-free, with support coming from an loosely-moderated online forum. The stark difference is likely to leave a lasting impression on students.

Example projects in the former category include:

- Extending a base class so that any `Enemy`, `Goodie`, `Destination` can appear or disappear according to a timer.
- Extending `Goodie` to cause a `Hero` to move in slow motion for a period of time.

A programmer who has experience with ALE could complete either project in less than ten minutes, writing less than 15 lines of code. However, a new user might take an hour to read enough code to know where to begin, and would need to think carefully about the object hierarchy and interaction routines to ensure a clean interface.

Example projects in the latter category include:

- Adding an obstacle that behaves like a pinball flipper
- Adding a "volcano" that spews `Goodies` or `Enemies`.

Neither of these projects would require more than 100 lines of code, but could likely take several hours of grappling with AndEngine.

5. LESSONS LEARNED

Many of our experiences with ALE confirm observations that others have made in the literature. Our informal experience strongly confirms the importance of keeping games violence-free and gender-neutral [3, 22]. This is not just a matter of preventing underrepresented students from losing interest: it also provides a challenge to violence-focused teams. Our requirement that games relate to social causes made it easy to emphasize the point, since students could not deny that they would never reach a large segment of their target market with violence.

The second critical lesson is that a framework like ALE must be bulletproof, and that ease of use trumps efficiency. The first version of ALE required students to manually register images by creating fields within their game class, computing image dimensions, allocating texture memory, and then loading image files into that memory. Though media registration is not critical code, it was the overwhelming source of bugs. Replacing this with less-efficient code that refers to media by filename virtually eliminated crashes in student code.

Third, we were amazed at the variance in student behavior with regard to modifying ALE. Some seniors were reluctant to change our code, while some first-year non-majors showed no hesitation. Certain design decisions, such as disabling objects instead of reclaiming them, proved valuable since novice students would modify ALE without understanding object lifecycles. Forbidding novices from modifying ALE is a dangerous proposition, since students should be encouraged to explore, regardless of their background. However, it helps that it is hard for students to break ALE.

Fourth, forbidding the use of copyrighted material is essential. The first time we used ALE, every project had copyrighted material. These students could not market their games. However, as we grew more strict, creativity blossomed. Engineers drew animation sequences, wrote music, and recorded their own sound effects. For our upper-level mobile programming class, we paired teams of students with "consultants" who were actually Graphic Design majors registered for an independent study in their home department. The quality of the artwork was exceptional, and the designers were graded and supervised by an expert in their field, rather than a computer scientist. By coordinating with an independent study, we were able to overcome some of the administrative challenges discussed by Wolz et al. [28]. Furthermore, we made it explicit that the artists were on equal footing: CS students who made last-minute requests for artwork changes, or who did not work with their consultant early to specify their needs, received no sympathy from the artists or instructor.

The main lesson for designers of new projects is that up-front work and a continuing effort to nurture creativity are essential. If ALE did not have significant documentation and thousands of lines of comments, the project would have failed. Furthermore, during our first trial with ALE, we would field requests from students after each lab, and then upgrade the framework for the following week. This produced more than a dozen new features during the course of the semester, but enabled every student to realize their game, even if they had no programming background. It also taught the students about how to specify requirements, and helped us converge toward a robust framework.

Finally, and perhaps most importantly, our experience shows that students can write compelling mobile games in under 18 hours, even without CS background. They quickly learned how to bend interfaces to overcome seeming deficiencies in a library (the most popular example was the use of an "invisible enemy" as a catch-all for terminating a level). Even when programming skills themselves were not a learning objective, students saw that programming could be fun, while also reinforcing attention to detail, clearly articulating specifications, and entrepreneurship.

6. CONCLUSIONS AND FUTURE WORK

Our experiences developing ALE and using it in CS projects has thus far been extremely positive. By emphasizing code reading as a core skill, we have helped students to enjoy programming while producing visually impressive, fun, and socially relevant games for smartphones. ALE will be released as an open source project in December of 2012, once we have completed testing on the latest version, which is being used by 96 first-year students this fall.

In addition to the experiences we have shared in this paper, we believe that there are many more opportunities for researchers and educators alike. Among the most exciting are the following:

- ALE can serve as the foundation for campus-wide programming competitions. Since the barrier to entry is low, and since the need for rich media often trumps the quality of the source code, it is possible that non-majors and new majors can "win" despite the greater programming experience of more senior competitors. This project can also encourage campus-wide engagement, since all students, staff, and faculty can participate in judging.
- We also believe that ALE is suitable for middle-school and

high-school outreach. While ALE requires students to write Java, many styles of game can be written without anything more than declarative code.

- We are also hopeful that researchers can formally evaluate the effectiveness of ALE in broadening participation and improving student outcomes and retention.

The key point, however, remains that by emphasizing reading code, a complex project like writing Java-based games for Android smartphones becomes tractable and enjoyable even for students without programming experience. As a project with tunable complexity and low administrative overhead, ALE is suitable for use as a capstone project in a non-ALE course, as a means of teaching software introduction, as an introduction to HCI issues related to smartphones, or as a fun interdisciplinary way to increase excitement and broaden participation in CS.

7. REFERENCES

[1] J. Adams and R. Pruim. Computing For STEM Majors: Enhancing Non CS Majors. Computing Skills. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, Raleigh, NC, Feb. 2012.

[2] J. Andrus and J. Nieh. Teaching Operating Systems Using Android. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, Raleigh, NC, Feb. 2012.

[3] L. Cassel. Personal Conversation, Dec. 2011.

[4] M. desJardins and M. Littman. Broadening Student Enthusiasm for Computer Science with a Great Insights Course. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, Milwaukee, WI, Mar. 2010.

[5] P. Dickson. Experiences Building A College Video Game Design Course. *Journal of Computing in Small Colleges*, 25(6):104–110, 2010.

[6] P. Dickson. Cabana: A Cross-platform Mobile Development System. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, Raleigh, NC, Feb. 2012.

[7] M. Goadrich and M. Rogers. Smart Smartphone Development: iOS versus Android. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, Mar. 2011.

[8] N. Gramlich. AndEngine – Free Android 2d OpenGL Game Engine, 2012. http://www.andengine.org/.

[9] I. Greenberg, D. Kumar, and D. Xu. Creative Coding and Visual Portfolios for CS1. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, Raleigh, NC, Feb. 2012.

[10] P. Hubwieser and M. Berges. Minimally Invasive Programming Courses - Learning OOP With(out) Instruction. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, Mar. 2011.

[11] S. Kurkovsky. Engaging Students through Mobile Game Development. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, Chattanooga, TN, Mar. 2009.

[12] O. Laadan, J. Nieh, and N. Viennot. Teaching Operating Systems Using Virtual Appliances and Distributed Version Control. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, Milwaukee, WI, Mar. 2010.

[13] S. Loveland. Human Computer Interaction That Reaches Beyond Desktop Applications. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, Mar. 2011.

[14] D. Malan. Reinventing CS50. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, Milwaukee, WI, Mar. 2010.

[15] R. McFall and M. DeJongh. Increasing Engagement and Enrollment in Breadth-First Introductory Courses Using Authentic Computing Tasks. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, Mar. 2011.

[16] V. Pauca and R. Guy. Mobile Apps for the Greater Good: A Socially Relevant Approach to Software Engineering. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, Raleigh, NC, Feb. 2012.

[17] C. Rader, D. Hakkarinen, B. Moskal, and K. Hellman. Exploring the Appeal of Socially Relevant Computing: Are Students Interested in Socially Relevant Problems? In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, Mar. 2011.

[18] D. Riley. Using Mobile Phone Programming to Teach Java and Advanced Programming to Computer Scientists. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, Raleigh, NC, Feb. 2012.

[19] D. Schuster. CS1, Arcade Games and the Free Java Book. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, Milwaukee, WI, Mar. 2010.

[20] G. Smith and A. Sullivan. The Five Year Evolution of a Game Programming Course. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, Raleigh, NC, Feb. 2012.

[21] K. Stolee and T. Fristoe. Expressing Computer Science Concepts Through Kodu Game Lab. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, Mar. 2011.

[22] K. Sung. Computer Games and Traditional CS Courses. *Communications of the ACM*, 52(12):75–78, 2009.

[23] tvtropes.org. Wax On, Wax Off, 2012. http://tvtropes.org/pmwiki/pmwiki.php/Main/WaxOnWaxOff.

[24] T. VanDeGrift, T. Caruso, N. Hill, and B. Simon. Experience Report: Getting Novice Programmers to THINK about Improving their Software Development Process. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, Mar. 2011.

[25] A. Vihavainen, M. Paksula, and M. Luukkainen. Extreme Apprenticeship Method in Teaching Programming for Beginners. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, Mar. 2011.

[26] D. Webb, A. Repenning, and K. H. Koh. Toward an Emergent Theory of Broadening Participation in Computer Science Education. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, Raleigh, NC, Feb. 2012.

[27] D. Wolber. App Inventor and Real-World Motivation. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, Mar. 2011.

[28] U. Wolz, L. Cassel, T. Way, and K. Pearson. Cooperative Expertise For Multidisciplinary Computing. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, Mar. 2011.