Using Video Games to Teach Introductory Computer Science Classes

Alexander Gunness
University of Minnesota, Morris
gunn0056@morris.umn.edu

ABSTRACT

Teaching introductory computer science is a notoriously difficult task. One proposed alternate method is to have students play video games. While implementation research is inconclusive, Applied-Behavior Analysis guidelines exist on how to create games for this purpose, and are used to analyze three examples that cover recursion circuitry.

Keywords

Teaching, Introduction Courses, Video Games, Recursion

1. INTRODUCTION

Teaching introductory computer science is a notoriously difficult task, and it is primarily during the these courses that students leave the Computer Science major, upwards of 40% [1]. Because of this, alternate methods of teaching have been looked into, and one method is having students play video games to learn concepts. Most of the research in the field has been inconclusive in showing benefits of this approach [2]. Even so, there are still some examples worth mentioning. Two sets of guidelines exist on how to make a video game as a teaching tool: one following successful game characteristics, and the other uses Applied Behavior Analysis [2].

2. BACKGROUND

The goals of introductory computer science classes are largely straightforward. The first and foremost goal is that introductory classes should set students up for classes later in the curriculum. Specifically, the classes should include some form of recursion or non-recursive iteration, basic methods on how to go about solving problems, an introduction to some kind of coding syntax is often included, and other basic concepts (like variables). The classes also should spark an interest of later Computer Science topics in the students while giving them a scope of what it has to offer.

UMM CSci Senior Seminar Conference, April 2014 Morris, MN.

Specifically how one goes about introducing concepts is a matter of debate, as is seen in ACM's document for guidelines on teaching introductory classes. There is no consensus on how introductory concepts should be taught, as some think teaching functional before object-oriented is better, while others think the opposite. Some think it is better for students to learn on non-standard devices like the Raspberry Pi, mobile devices, game devices, and others. What this means in regards to video games is that because there is no method recognized as being better than others, they can be used and still adhere to the ACM standards. [3]

2.1 Difficulties in Learning Recursion

One of the important topics to learn in an introductory class is recursion, a notoriously difficult topic to teach [1, 5]. Part of the issue is that to understand recursion one needs to understand how the program stack works. It is also easy to confuse recursion with non-recursive iteration, as in many simple examples both can appear to essentially be the same [5].

A series of examples of this issue can be seen in the following code snippits (modified from [5]). Imagine an introductory student is trying to comprehend R1.

```
public void R1(int x) {
  if (x > 0){
    System.out.print(x);
    R1(x - 1);
  }
}
```

This hypothetical student could think R1 is the exact same as I1.

```
public void I1(int x) {
  for(int i = x; i > 0; i--){
    System.out.print(i);
  }
}
```

I1 gives the same output as R1, but is using iteration instead of recursion. Both of these, in some sense, say "Print x, then print x-1 until x=0". Due to this it can be hard to conceptually understand what is different between the two (such that recursion can cause a stack overflow while iteration cannot, as an example).

Without fully understanding the program stack R2 could appear to be the same as R1.

```
public void R2(int x) {
```

```
  if (x > 0){
    R2(x - 1);
    System.out.print (x);
  }
}
```

While this is also recursion, the output is in the opposite order than R1's output. If the student does not fully understand that the print occurs while the stack is emptying itself they might think it prints prior to the recursive call finishing.

Without understanding the differences between R1, I1 and R2, the more complicated examples, such as working with binary trees, are much harder to understand.

2.2 Why Video Games

Video games are a unique kind of entertainment. In order to play a game well one must learn the game's system and figure out what limited actions are possible within the system while succeeding in challenges brought forth by the game. This means that they inherently require the player to learn concepts and ways of thinking in order to succeed. [2]

One example of this is from the popular games Portal and Portal 2, where a common phrase is "now you're thinking with portals". In those games one is given a device that creates two portals that effectively teleport the player from one portal to the other. The portals are used in a myriad of ways to solve puzzles (such as putting one portal on the ground beneath the other, which is on the ceiling, in order to get a large amount of velocity from gravity). In these games one has to wrap their mind around this special ability and its limitations in order to get to the exit of each level.

This is notably similar to learning Computer Science. One must learn how a language works and use available commands to their advantage to overcome coding problems. If a video game is structured correctly (more on that in the next section) it can theoretically be a decent vehicle for learning Computer Science.

Video games have a second compelling rationale: they are inherently motivating [1, 2]. The reason the video game market is succeeding is that they are made to be fun and enjoyable.

Since video games inherently educate and motivate while being able to invoke Computer Science-style thinking they are theoretically a good tool for teaching computer science [2].

3. DESIGNING A GAME FOR TEACHING

Games should theoretically have better inherent motivation than standard lesson methods. But there is more to it than "student plays video game, student learns topic".

There are notions that "fun, flow, engagement, feedback, goals, problem solving, game balance and pacing, interesting choices, and fantasy narrative, among many others" are all needed in order for a video game to be successful, and by extension, facilitate learning [2].

There is one tested element a video game needs in order to be educational. This one element is intuitive: if tasks in the game are completable without the player learning anything then the game taught the player nothing. This same element also makes players more motivated to keep playing. [2]

One thing to keep note of is that while the goal of an educational video game is to make it educational, it is a mistake to let the educational aspect hurt the gameplay. That said, the learning aspect must be integral to a game's mechanics rather than being an afterthought. [2]

[2] has a list of seven common traits of successful commercial games (though this list is not exhaustive).

- Short, medium, and long-term goals are present throughout the game
- Decision-making is usually required in order to meet said goals
- Games provide "immediate, appropriate, and specific feedback to players"
- There is a complex way of giving rewards
- Long tasks are usually broken into shorter ones. The shorter tasks are each taught separately before they are used together
- A player is required to be good at a specific task before being allowed to attempt harder tasks
- There are multiple ways to solve a problem and none of them are obvious, while there are obviously wrong methods

While these seven traits are needed to make a commercial game successful they are also what are needed to make a game enjoyable. The motivational nature of video games is directly related to their joy factor, so this list should also be adhered to when making an educational game.

3.1 Applied Behavior Analysis (ABA)

Of the seven successful commercial game traits the most important is the immediate and specific feedback video games have to offer and their ability to change feedback on a per-player basis. Applied Behavior Analysis (ABA) is an educational framework based on this concept, so it is a useful framework to work from when designing educational games. ABA and video games also share the trait that the students/players have specific goals to achieve, time constraints, rewards for meeting said goals, and are given constant feedback.

ABA is a teaching style in which the lecture format is rejected, and instead the teacher works with each student individually. Feedback is given often and per-person, and a student is supposed to continue at a topic until they have at least a 90% understanding [2].

ABA has been very successful wherever it has been implemented (ranging from elementary school to high school). However, ABA has recurring problems that make it hard to implement. First, it requires a teacher put in a large amount of time training and planning. Second, there of a bias against having students constantly repeat efforts on the same topic. [2].

Video games can stick to ABA standards and get around the issues ABA is normally plagued by. Far less is required of the teacher because video games can teach players on their own. Players are also more likely to keep at a problem until they understand it, as games are motivating.

3.2 ABA Steps to Designing an Educational Game

[2] lays out three main steps to designing a game to adhere to ABA standards: defining/measuring desired behavior, recording/analyzing changes in behavior, and giving feedback. These three steps then need to be adapted to each student.

The first step when designing a game is to define what skill/behavior is supposed to be improved. Those skills should be the same as the skills used in the gameplay, not just similar ones. Measuring a student's understanding should incorporate both their accuracy and speed of finding correct answers.

The second step regards changes in behavior. Behavior should be recorded by logging actions the player takes that fall within the important skill's scope. For example, one would not record the player's choice of hair color if the goal is to get the player to understand recursion. The correctness/speed of relevant behavior would be automatically analyzed by the game.

The third step is feedback. Different kinds of feedback, derived from the analysis of the player's actions, would be given to each player. Role playing games are a good example, as experience points are rewarded for "correct" actions (thus positive reinforcement) or items/experience can be taken away upon failing (typically dying).

Each person is different, and different kinds of feedback motivate different kinds of people. Ideally the game would detect what motivates a player and act accordingly. This requires recording the tasks the player spends more time on and what kind of rewards those tasks give. When and how rewards are given out is also an issue, as they cannot simply be given out after each task is complete. Usually the rewards have variable triggers and variable difficulty in acquiring and typically rewards come less frequent as the game progresses. This can be seen in most MMOs (massive multiplayer online games) and Facebook/mobile games.

The overall difficulty should also be variable, as to allow players to start where they are comfortable and grow from where is natural for them.

4. RESEARCH EXAMPLES

Most of the research done in this topic stems from good ideas but often lack concrete evidence that their approaches are actually working [2]. That said, it would be useful to look at some examples of research that has been done. Below are three examples of research in the area.

4.1 Circuitry

The driving force behind the study presented in [4] was the noted drop in students' ability to pay attention and willingness to complete assigned readings.

The game (currently unnamed) takes place in a 3D environment. The end goal is to reach the exit via opening doors and unlocking skill upgrades. Each door can only be unlocked by solving a logic gate problem, and the game switches to a 2D environment when attempting to do so (see Figure 1). The player is provided something equivalent to a truth table (see the top of Figure 1), along with inputs on the left-hand side and the outputs on the right. The puzzles are solved by drag and dropping gates onto the board and wiring inputs, gates, and outputs together via mouse clicks. The answer is constantly evaluated and correctness is displayed just beneath the truth table. Players have the option to toggle the inputs on and off so that they are capable of debugging their answer and figuring out the correct one. The two skill upgrades are unlocked via a puzzle of this same style. As per normal game standards, the circuits start off easy and progress in difficulty.

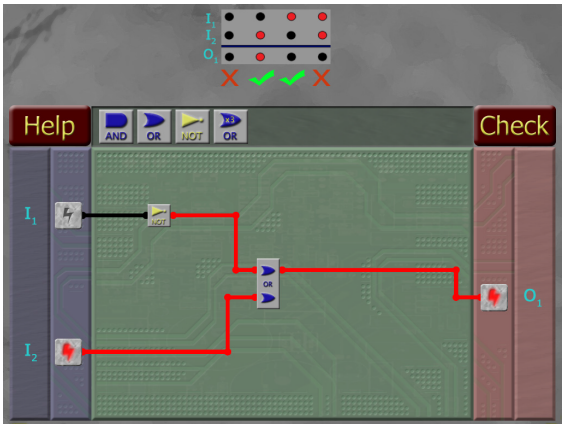

Figure 1: The circuit game's 2D interface, [4]

The target group for the game was students who were previously or currently enrolled in an introductory class that covered circuitry/truth tables.

In 2007 the preliminary test was conducted using thirteen students from the target group. The study primarily was concerned with feedback on the 2D interface and what students thought of using video games to learn. More than 60% preferred video games over paper problems [4].

In 2008 a pilot test took place involving nine students from the target group and four students who were not. In this study the students were capped at 75 minutes of playtime. After playing they took a survey that assessed their opinions of the game and demographic details. The feedback was generally positive and students appeared to enjoy the game. Some of them opted to continue playing beyond the capped time in order to reach the exit. Students appeared to enjoy the sense of achievement, ability to have immediate feedback, and that the game was generally fun.

Before and after playing the nine test students were given a conceptual test of two circuit problems to solve. On the pretest eight of them got 0% while one got 100%. While the students played it was noticed that they had more of a guess-and-check tactic while later on they were more methodological. On the post-test, of the eight who originally scored 0%, four of them still scored 0% while four got a higher score (exact numbers were not listed).

Though the test group's improvement seemed small the authors felt that, given how short the test was, it was still promising. They also noted that students had last seen the topic either a month or semester prior, and only for 1.5 weeks. In the future the authors plan to conduct a larger study, add a plot, more variety to the puzzles, a tutorial, and a system to hand out context-sensitive hints to players who are stuck on problems. [4]

Of the seven traits from [2] this game was successful at having a long-term goal, "immediate, appropriate, and spe-

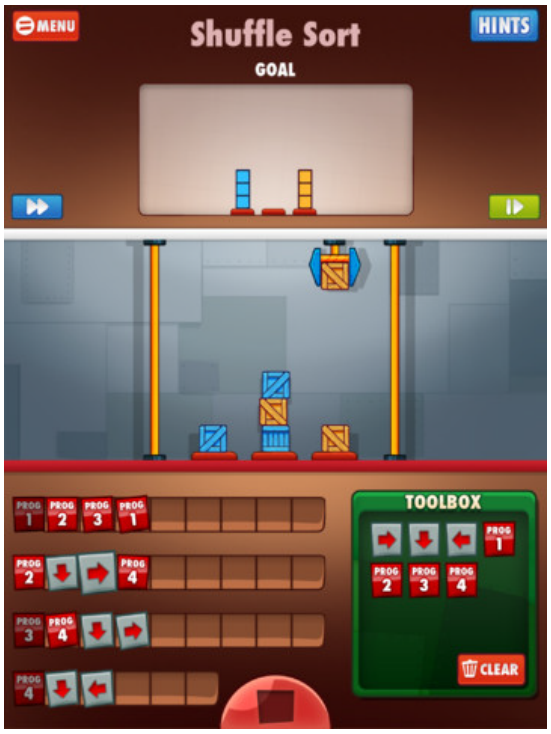

Figure 2: Commercial Cargo-Bot's GUI, [5]

cific feedback", and being required to complete easier tasks before harder ones. It was also successful at knowing what behavior was desired. The future work proposed will help the game give feedback on a per-player basis, based on the actions the players are taking. With the proposed updates the game will better match the ABA guidelines given in [2]'s.

4.2 Recursion with CargoBot

Recursion is notoriously difficult to teach, and is a fundamental concept that is necessary to understand in order to do well in Computer Science. One issue with teaching recursion is that concepts are easier to understand if they can be likened back to past knowledge, and there are few instances of recursion in everyday life [5]. This was the line of thinking behind this study.

Cargo-Bot is a commercially available game for Apple's iPad and was not designed with education in mind. The authors of this study used it with permission. In the game, players use a simple visual language in order to issue commands to a robotic arm. The goal is to take a given start state and transform it into an end state by moving the colored boxes around. The game supports a construct very akin to recursion but does not support looping in any other fashion. In the game players are allowed to define four procedures, which carry out a set of commands. The first three are allowed eight commands while the last is only granted five. The commands available are down (which places/picks up boxes), left, right, procedure 1, procedure 2, procedure 3 and procedure 4. If clauses that check whether a box is currently in the arm or the color of the box are also available. When the player hits the start button procedure 1 is carried out, and the other ones are only run when called by the appropriate command. The play button turns into a stop button that resets the state of the game (saving the commands within the procedures), which allows players to change their procedures to re-attempt the puzzle. This basic language fully supports self-referencing, each successive recursive call does act to get the state of the game closer to the end state, and supports something akin to a base case (the end state).

For the study Cargo-Bot was modified to allow players to attempt levels at will instead of the linear fashion originally intended by the game's creators.

The experiment had two groups: the "control" group (21 students) and the "experimental" group (26 students). The students involved in the two groups were all from the same school taking the same AP Computer Science class (different sections). Both classes followed the same schedule and neither had been introduced to recursion prior to the study.

Each group played Cargo-Bot, had a lecture, and had three tests. The ordering for these events was different for the two groups. The control group had a 20 minute pre-test, 50 minute lecture, and a 15 minute mid-test on day one. Day two included playing Cargo-Bot for 90 minutes and a 20 minute post-test. The experimental group had the pre-test and played Cargo-Bot on day one, while taking the mid-test, lecture, and post-test day two.

The lecture covered defining recursion, the stack, and included examples of recursion. Some of the examples mimicked Cargo-Bot style puzzles.

Each of the tests included both trying to understand a recursive function and writing their own. Their understanding of existing recursive functions was done by being asked what a specific call of it returns (such as `Fibonacci(10)`). The given function and the one they had to write became harder on each successive test.

For the portion where students wrote their own recursive functions the control group had a drop in test scores after the lecture, but had a large increase after playing Cargo-Bot. The experimental group had a large increase after playing Cargo-Bot but only a small increase after the lecture. For the portion where students looked at existing recursive functions both groups did worse on the mid-test than the pre-test, then did better on the post-test than the pre-test. This showed that while playing Cargo-Bot did increase their ability to write recursive functions, it did not help their ability to understand existing recursive functions. The authors note that this makes complete sense as Cargo-Bot only has players creating procedures.

The authors originally theorized that having a context for recursion makes understanding it easier. This study did not prove that, but it did show that it is preferable to have the exploration take place prior to the lecture.

In the future the authors plan to test this method of teaching recursion with college students, and plan to change how the game represents carrying out procedures to help students better understand how to follow a recursive function. They also plan on looking into using games to teach other topics such as threading. [5]

Of the seven traits from [2] the game did provide "immediate, appropriate, and specific feedback" to players. Players had the ability to jump around and do problems at-will, which directly goes against requiring players to be good at a specific task prior to advancing. Of the ABA guidelines it was capable of giving feedback in the form of immedi-

```
class Test {
  public void TreeTraversal() {
    Tree myTree = new Tree();
    depthFirstSearch(myTree.root);
  }

  public void depthFirstSearch(Node node) {
    Thought.moveTo(node);
    // Check for Base Case
    if ((node.returnRight() == null)
              && (node.returnLeft() == null)) {
      return;
    } else { // Recursive calls
      // Travel to node's right child
      if (node.returnRight() != null) {
        depthFirstSearch(node.returnRight());
        Thought.moveTo(node);
      }
      // Travel to node's left child
      if ([YOUR_CODE] != null) {
        [YOUR_CODE]
      }
    }
    return;
  }
}
```

Figure 3: Modified scaffolding code from EleMental's level 2, [1]

ately showing players what their procedures did correctly or incorrectly. The proposed improvements will help teach players recursion better, but will not help it follow any of the guidelines from [2].

4.3 Recursion with EleMental

EleMental is a cross between coding and playing a game that attempts to help teach recursion. This example focuses more on direct experience than an abstraction of recursion like CargoBot [1].

The game had three levels. The first had two simple tasks. Players wrote a "Hello World" script and manually walked their character through a binary tree in a depth-first search pattern. The game would give hints on how to correctly walk their character through the tree.

In the second level players were given most of the code for depth-first search (Figure 3) and were asked to finish the code. The only missing bit was how to handle traversing through the left-hand side. The scaffolding code was kept shorter to avoid code that was too complex for introductory students. At UNCC they taught C++ and Java but the game engine was written in C#, which is very similar to the other two, so the scaffolding code also included C# specific quirks that were not shared with C++ or Java. This allowed students to code without knowing C# specifically, except that `Console.WriteLine` was needed instead of `cout` or `System.out.println` in the "Hello World" level.

If the student wrote code that did not compile, an error message popped up that included what line caused the error. If the code did compile but did not match the intended pattern a custom error message popped up explaining where the inaccuracy was.

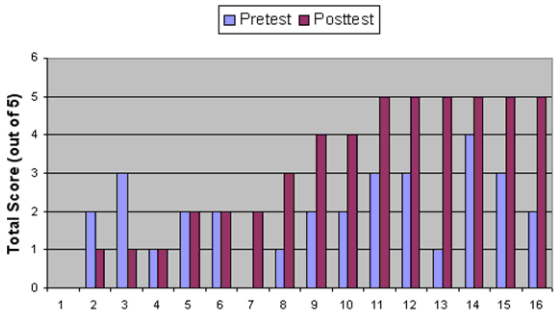

Figure 4: Pre- and post-test EleMental scores, [1]

After level 2 was solved the game showed a character walking through the tree and explained what was happening without mentioning the program stack, but spoke about how their character would not go backwards on the tree without first having "talked" to both of the current node's children.

Level 3 finally introduced the program stack. Students again were given scaffolding code, but this time were required to write the left and right-hand side of depth-first search. The reasoning behind using the same exact code (minus a few lines) from level 2 was never given. The possibility of students simply copying from memory what they had seen not a few minutes prior without really understanding it was also never discussed. After level 3 was solved students had to walk through the tree "using the keyboard and mouse" (how this is different from what was done in level 1 is not explained) while the game gave a visual metaphor for the stack.

There was also a metaphor using telephones during level 3. The example, in brief, went "A asks B a question, B asks C, C asks D, then E. D and E each give C an answer, who tells B, who then tells A." This was intended to help explain how the program stack and recursion work.

There was a single playtest of EleMental. Prior to playing a pretest was given judging how well players knew recursion going in, while a post-test was given afterwards. There were 43 participants that took the post-test, but only 16 of them took the pre-test. Of the 16 there was "1 freshman, 1 sophomore, 10 juniors, 3 seniors, and 1 post Baccalaureate student. Fourteen were in computer science-related majors, 1 was in computer engineering, and 1 was in graphic design with a minor in computer science." The participants all had already completed or were enrolled in Data Structures and Algorithms. They also noted how many hours a day each person played video games and what type of gamer they considered themselves, but due to the small test size this information was not used.

Figure 4 shows the pre- and post-test scores for the 16 students. In [1] statistical analysis was done that showed a significant increase in test scores after playing EleMental. Notably, there were no significant correlations between time spent in the game and post-test scores.

All 43 students also completed a survey of the game. Many questions were answered with one of five options ranging from "strongly agree" to "strongly disagree".

- 34 "strongly agreed or agreed that they enjoyed playing the game."
- 35 "agreed that they enjoyed creating and compiling

Response Categories	By respondents	By categories
In-Game Coding	29% (12 of 42)	18% (12 of 67)
Visualization	62% (26 of 42)	39% (26 of 67)
Education	38% (16 of 42)	24% (16 of 67)
Hints	19% (8 of 42)	12% (8 of 67)
Game Play	11% (5 of 42)	7% (5 of 67)

Figure 5: Participant's favored aspects of EleMental, [1]

their own code inside the game"

- 34 "agreed the game was helpful in learning computer science concepts"
- 34 "students agreed that the second level with the AI walkthrough was more helpful in learning depth-first search than the other two levels"

The players were also asked what they liked most about the game, and were allowed to select multiple options. The visualization of recursion was the overall favorite.

In the future the research group plans on showing the advantages of using recursion. Part of this will be by replacing level 1's "Hello World" with a "brute-force program to perform tree traversal". Each level will also show what incorrect compilable code does, as to better allow players to understand what they are doing wrong. Level 3's stack and telephone metaphors will be improved as the UNCC professor who teaches Data Structures and Algorithms noted this was the hardest part for students taking the class to understand.

They also plan on adding an experience point (XP) system to the game to add further motivation to the game. This is something many games implement to get players to play longer. Interestingly, this works even if the XP does absolutely nothing, as gamers simply like getting to higher levels and getting more XP.

UNCC also plans on integrating EleMental into their Data Structures and Algorithms class to get further testing and to allow their students "the benefit of an alternative form of learning" [1].

The game was designed to teach recursion, though among the testers was 10 juniors, 3 seniors, and a post Baccalaureate student. This means that either UNCC did not teach recursion prior or the test was done on studnets who already knew recursion. In the latter case the results do not mean that EleMental is "good" at teaching recursion, but rather is good as a refresher.

Of the seven successful game traits from [2] EleMental had some level of "immediate, appropriate, and specific feedback" and being required to complete easier tasks before harder ones. The game was also designed knowing what behavior was desired. The future work proposed will help the game give better feedback for understanding why incorrect code is incorrect. EleMental does go against one of the seven successful game traits, however, as there was only one correct way to solve a problem.

5. CONCLUSIONS

It is still debatable whether or not there is room for video games within introductory computer science education. There is no conclusive research showing video games are a better or equal alternative to traditional teaching methods. Interestingly, even though there are guidelines for creating successful educational games, commercial games can potentially be tapped as a resource as well. The ABA guidelines for designing a game are as follows: the game needs a way of defining/measuring desired behavior, recording/analyzing changes in behavior, giving feedback, and changing to be player skill dependent [2]. The three examples shown did not have a way of satisfying the second or fourth guidelines.

Ultimately this means that much more research needs to be done on the actual effectiveness of video games in teaching computer science.

6. FUTURE WORK

As mentioned most of the research in this field is sub-par. This is because there is little research in using video games in this manner (as most video game-related computer science teaching is done through students creating their own games). There is also no conclusive research on whether or not video games are actually better than usual teaching styles, only that video games can work [2].

In light of this what needs to happen is games need to be designed as per [2]'s two sets of guidelines, and better research needs to be done. The research should compare teaching using video games to not using video games, and the sample sizes should be larger than the current ones. This will help decide whether or not video games are actually more helpful, or if they are helpful at all.

7. REFERENCES

[1] A. Chaffin, K. Doran, D. Hicks, and T. Barnes. Experimental evaluation of teaching recursion in a video game. In *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, Sandbox '09, pages 79–86, New York, NY, USA, 2009. ACM.

[2] C. Linehan, B. Kirman, S. Lawson, and G. Chan. Practical, appropriate, empirically-validated guidelines for designing educational games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1979–1988, New York, NY, USA, 2011. ACM.

[3] M. Sahami, A. Danyluk, S. Fincher, K. Fisher, D. Grossman, E. Hawthorne, R. Katz, R. LeBlanc, D. Reed, S. Roach, E. Caudros-Vargas, R. Dodge, R. France, A. Kumar, B. Robinson, R. Seker, and A. Thompson. Introductory courses. In *Computer Science Curricula 2013*, Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, pages 39–45. ACM and IEEE, 2013.

[4] V. Srinivasan, K. Butler-Purry, and S. Pedersen. Using video games to enhance learning in digital systems. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, Future Play '08, pages 196–199, New York, NY, USA, 2008. ACM.

[5] J. Tessler, B. Beth, and C. Lin. Using cargo-bot to provide contextualized learning of recursion. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, ICER '13, pages 161–168, New York, NY, USA, 2013. ACM.