Using Cargo-Bot to Provide Contextualized Learning of Recursion

Joe Tessler
University of Texas at Austin
joe.r.tessler@utexas.edu

Bradley Beth
University of Texas at Austin
bbeth@cs.utexas.edu

Calvin Lin
University of Texas at Austin
lin@cs.utexas.edu

ABSTRACT

This paper presents a new method of teaching recursion in which students first play a video game to contextualize recursive operations. Results from a controlled experiment with 47 high school students taking AP Computer Science A indicate that this instructional strategy produces significant improvements in students' understanding of recursion. Additionally, survey results show that nearly every student enjoys the learning activity and is confident in his or her ability to accomplish the recursive exercises.

Categories and Subject Descriptors

D.3.3 [**Language Constructs and Features**]: Recursion; K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms

Human Factors

Keywords

Education; Recursion; Video Games

1. INTRODUCTION

Recursion is an important computer science concept, but one that is notoriously difficult to learn [14]. Students sometimes fail to recognize the distinctions among different invocations of the same function, and they often get confused by the bookkeeping required for each recursive call. Specifically, research shows that students struggle with the unfamiliarity of recursive activities [1], the visualization of the program execution [9], the backward flow of control after reaching the base case (i.e., passive control flow) [19], the comparison to loop structures [1], and the lack of everyday analogies [16].

The considerable body of previous work in teaching recursion includes conceptual and abstract discussions of recursion and its control flow [5, 8, 17, 19, 20, 26]; comparisons to other topics and disciplines [6, 12, 18, 23]; and the use of visual aids and hands-on activities [2, 7, 10, 11, 21, 24]. Gunion et al. [10] claim that we need further "efforts to find useful and effective instructional approaches". Unfortunately, although previous research does include some studies that involve in-class experiments [2, 8, 19, 21, 24], none of the previous research includes controlled experiments that provide statistically significant evidence that the proposed teaching methods improve student learning of recursion. This paper attempts to address this shortcoming.

Our work builds on the basic notion of contextualized learning, which suggests that students learn best when they can relate new concepts to previously understood concepts [3, 4, 25]. While there exist few natural instances of recursion in most students' lives, there is a growing number of computer games that require the player to think recursively. For example, in both Cargo-Bot and Light-Bot 2.0, players control virtual robots by creating programs using a simple visual language. In Cargo-Bot, the goal is to control a robotic arm so that it moves a set of crates to a specified goal configuration. Notably, Cargo-Bot supports recursion but no explicit looping structures.

It is natural to wonder if students who play Cargo-Bot are able to transfer their experiences thinking recursively to new contexts (e.g., Java). Thus, we investigate a new method of teaching recursion in which students play Cargo-Bot to situate learning before they are formally taught recursion.

There are several reasons to suggest that this approach might improve student learning:

- Cargo-Bot contextualizes recursive operations in concrete terms of crates and cranes, rather than traditional examples that rely on mathematical abstractions. The use of Cargo-Bot examples during direct instruction connects instruction with students' prior experiences within the game environment.
- As with most video games, Cargo-Bot is fun and addictive, so students are likely to be motivated to explore the game independent of its instructional use.
- Cargo-Bot provides multiple opportunities for students to practice thinking recursively, as they are given progressively more difficult problems to solve. Thus, students are able to work through a number of examples at their own pace.

In this paper, we describe our new approach to teaching recursion, and we evaluate our idea using an experiment conducted with 47 students across two AP Computer Science

ICER'13, August 12–14, 2013, San Diego, California, USA.

http://dx.doi.org/10.1145/2493394.2493411 ...$15.00.

A high school classes. One class serves as a control group while the other serves as the experimental group. The experimental group plays Cargo-Bot for an hour before learning about recursion. In contrast, the control group first receives a lecture on recursion; they then simply play Cargo-Bot. Assessments are given to each group at each point in the process to measure learning gains.

We find that the experimental group experiences a significantly greater increase in assessment scores from playing the game than the control group does from direct instruction alone. Moreover, both the control and experimental groups experience the greatest increase in scores after playing the game.

The remainder of this paper is organized as follows. Section 2 defines recursion and explains why it traditionally has been difficult to learn. Section 3 places our work in the context of prior work. Section 4, describes our experimental design, including Cargo-Bot, and Section 5 evaluates our solution, before we conclude.

2. BACKGROUND

In this section, we briefly define recursion and present common misconceptions that students develop when first introduced to the topic.

Recursion is a powerful programming tool that elegantly solves many complex problems. A recursive function is one that either directly or indirectly makes a call to itself, typically defined in terms of a smaller instance of itself, [22] as shown in Figure 1. In our experiment, we concentrate on simple functional recursion, which is typically taught to novice students in introductory courses. Thus, we do not discuss the distinctions among recursive data types, the differences between generative and structural recursion, or ties to formal languages. Every recursive function should obey the following rules:

1. The function must include a means for *terminating* with at least one base case that can be solved without using recursion.
2. The function should rely on a *recurrence relation* to compute at scale.
3. Successive recursive calls must progress toward a base case.

Students typically struggle with recursion because it requires a mental model of the program stack to recognize the backward flow of control after reaching a base case [9, 19]. For example, students may incorrectly conclude that the functions in Figures 1(a) and 1(b) produce the same output. However, with a proper understanding of the program stack, it is clear that `f1` will output `x` before the recursive call, which then outputs `x - 1`, so `f1` prints the numbers in descending order. By contrast, each invocation of `f2` is pushed on the stack before anything is printed; after reaching the base case, `f2` outputs `0`, and then there is a backward flow of control through each invocation of `f2`. So `f2` prints the numbers in ascending order.

Beyond this, students are sometimes confused by simple but contrived examples of recursion, such as the one in Figure 1, that have better iterative counterparts, because the students then incorrectly associate recursion with loop structures [1]. Furthermore, such examples fail to show students the benefits of recursion [16].

```
// Precondition: x > 0
public void f1(int x) {
  System.out.print(x);
  if (x > 0)
    f1(x - 1);
}
```

(a) Print before recursive call.

```
// Precondition: x > 0
public void f2(int x) {
  if (x > 0)
    f2(x - 1);
  System.out.print(x);
}
```

(b) Print after recursive call.

Figure 1: Two functions that highlight the flow of control in recursion.

3. RELATED WORK

There is a considerable body of research dedicated to improving the ways in which we teach recursion.

Ginat and Shifroni [8] found that "teaching recursion with an emphasis on the declarative, abstract, level of recursion considerably improves the student's ability". Similar to the work of Sooriamurthi [20], they claimed that focusing on what the recursive function should accomplish, rather than how it goes about doing it, is the "key" to comprehending recursion. Cargo-Bot requires students to focus on the end goal rather than the bookkeeping and procedural internals of the recursive process.

Edgington [5] suggested using the "concept of someone delegating a task to another person" as an example of a divide-and-conquer algorithm. Wirth [26] suggested asking students to solve an inherently recursive problem: How can they randomly parallel park cars on a city street? Given the lack of real-world examples, Wiedenbeck [23] advocated recursion analogies that come from programming. Our work incorporates these suggestions by introducing students to a video game that actually elicits recursive thinking.

Scholtz and Sanders [19] studied the use of tracing recursive methods as exercises for students. They found that "trace methods are essentially mechanical processes that can allow students with little understanding of recursion to correctly evaluate a recursive function but that students do not fully understand recursion and in particular have difficulties with the passive flow". They suggested the use of diverse recursive examples so that students can "learn recursion from different perspectives," and they claimed that students require sufficient practice designing their own recursive functions. Our use of video games gives students ample time to explore many recursive examples and to practice writing recursive functions in an interactive and visual manner.

A number of authors encourage the use of visual aids when teaching recursion. Hsin [11] developed a recursive graph that "can help students understand the flow of a recursion process". Similar to the work of George [7], Wilcocks and Sanders [24] use a program animator to assist students in "extrapolating a correct mental model of what recursion is". However, experiments by Stasko et al. [21] found no significant result suggesting that algorithm animators as-

sist learning. They suggested that future research focus on allowing students to construct their own animations. Our work directly addresses this recommendation, as students use a video game to create functional solutions and their accompanying animated visualizations.

Closely related to our work, Chaffin et al. [2] developed a video game that allows students to write depth-first search algorithms and interact with a visualization of a binary tree. They found that students achieve statistically significant learning gains after playing the game. However, their solution provides limited experiences with recursion, while Cargo-Bot offers many problems of various levels of difficulty in a fun and engaging environment.

4. EXPERIMENTAL DESIGN

The goal of our experiment is to determine whether Cargo-Bot can be used to improve the way that we teach recursion to novice computer science students. Specifically, we measure performance gains on a traditional code-based assessment over recursive routines before and after game play. We also measure the students' learning gains achieved by playing the game both prior to direct instruction and after. Thus, our experiment tracks two groups of students. One group, the experimental group, plays Cargo-Bot before receiving a lecture on recursion. The other group, the control group, receives the lecture on recursion and then plays Cargo-Bot.[1]

	Day One		
Control	Pre-Test	Lecture	Mid-Test
Experimental	Pre-Test	Cargo-Bot	

	Day Two		
Control	Cargo-Bot		Post-Test
Experimental	Mid-Test	Lecture	Post-Test

Table 1: Experimental time line for day one and two.

To measure the effects of the various teaching components, we give each group a series of tests, (1) a pre-test, which measures the students' initial facility with recursion, (2) a mid-test, which evaluates the effectiveness of one teaching component, and (3) a post-test, which measures the effectiveness of the second teaching component.

Students in both the control and experimental groups have 20 minutes to complete the pre-test, 15 minutes for the mid-test, and 20 minutes for the post-test. The lecture requires approximately 50 minutes, and students have 70 minutes of in-class time available to play Cargo-Bot. Accordingly, students utilize the entirety of each 90 minute class period even though the individual activities vary each day and between groups.

4.1 The Video Game

Cargo-Bot is a video game for the Apple iPad in which users "teach a robot how to move crates"[2] (see Figure 2) by writing recursive programs in a lightweight visual programming language. The game was not designed for educational use, and its developers are not directly involved in our research. Other recursion-based video games, such as LightBot 2.0[3], have similar traits and may be just as effective.

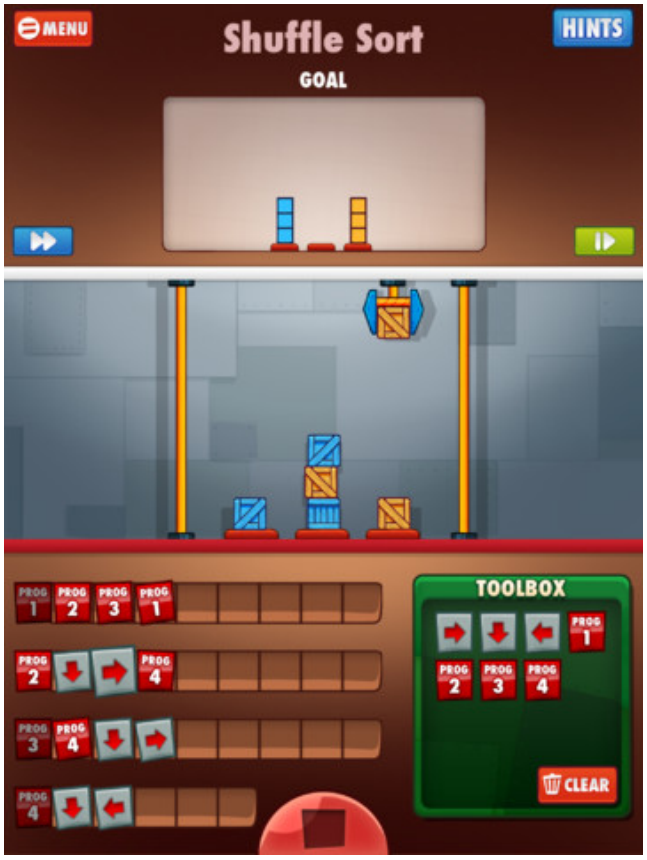


Figure 2: Screen shot of the original Cargo-Bot iPad game.

With the developers' permission, we rewrote Cargo-Bot in JavaScript to make it accessible to all students with Internet access. We also modified the game to include user identification and tracking, allowing us to associate game play with individual students. Our rendition includes all of the 36 levels and GUI components that are found in the original video game; our version is open source and available online[4].

Figure 3 shows our rendition of the game. The user-defined program in Figure 3c consists of five commands following the label `F0`: The first command tells the robot to move down, picking up or dropping any cargo that it might hold. The second command moves the crane right if the crane is currently holding any cargo, and the third command moves the crane right again if it is holding yellow cargo. The fourth command moves the crane left if it holds no cargo. Finally, the last command calls the function again to repeat the actions.

The 36 levels of game play are separated into six difficulty categories: Tutorial, Easy, Medium, Hard, Crazy, and Impossible. In the original game, players must complete one level before moving on to the next. We remove this restric-

[1]On day two the control group essentially becomes the experimental group, so the term "control group" is perhaps a misnomer.

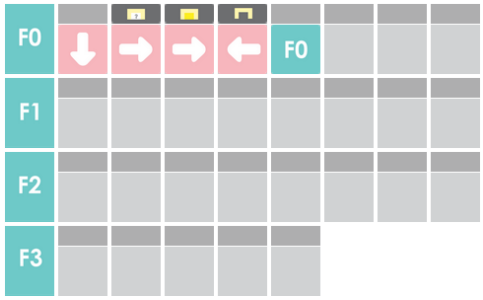
[2]`http://twolivesleft.com/CargoBot`

[3]`http://armorgames.com/play/6061/light-bot-20`

[4]`https://github.com/jtessler/cargo-bot`

(a) Goal configuration (top), animator window with initial configuration, and game controls (bottom).

(b) Available actions and conditionals.



(c) User-defined functions.

Figure 3: Various components of the Cargo-Bot GUI.

tion, allowing students to explore as many levels as they wish within the allotted time.

4.1.1 *Recursive Thinking in Cargo-Bot*

At first glance, it is not clear that Cargo-Bot programs utilize recursion. The simplest programs use tail recursion, which is indistinguishable from iteration given the language's visual nature. For example, Figure 3c shows a simple Cargo-Bot routine, `F0`, which iterates by calling itself, `F0`; however, it is not clear if the program is calling a new instance of `F0` or simply executing a `GOTO F0` command—in effect, interpreting `F0` as a label rather than as a recursive function.

Harder problems, however, require the user to make the nuanced distinction between the recursive and iterative interpretations of the code. For example, the solution in Figure 4 uses the program stack to store the number of leftward movements needed to return the crane to the leftmost platform. A non-recursive interpretation of the code fails to account for this implicitly stored value and would execute at most one leftward movement.

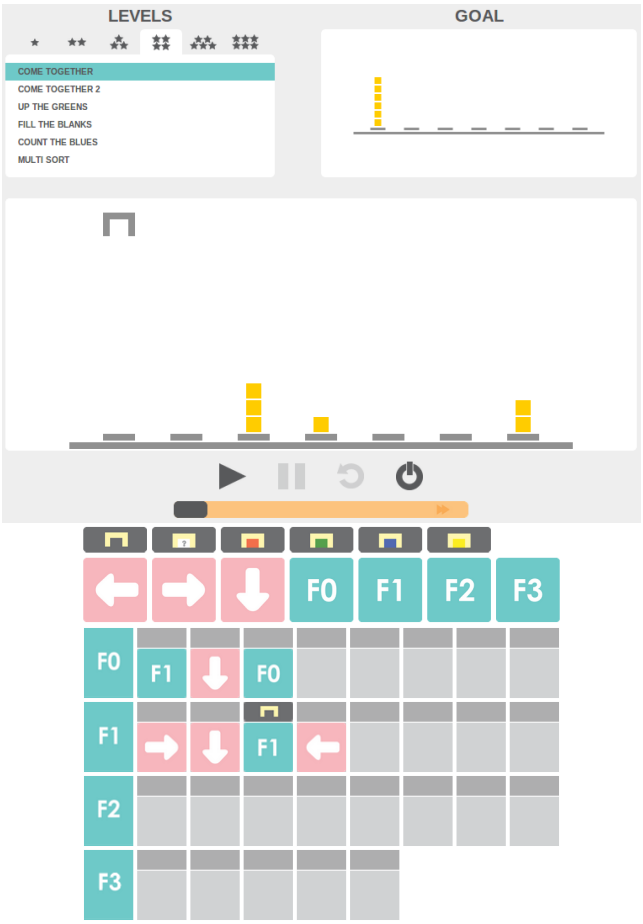


Figure 4: An advanced recursive solution requiring passive control flow.

4.2 Direct Instruction

Students in both the control and experimental groups receive the same material for an equivalent amount of time—50 minutes. The discussion begins with an introduction to recursion similar to the one presented in Section 2. After defining recursion, we provide a basic overview of the program stack and work through a series of tracing problems that exhibit the backward flow of control.

Examples mimicking Cargo-Bot scenarios are included to situate recursive operations within the concrete Cargo-Bot environment. We present recursion using examples that utilize the program stack, such as the one depicted in Figure 4. Additionally, Cargo-Bot examples are presented for comparison in Java-like syntax, with explicitly named methods representing each of the available actions in the Cargo-Bot visual programming language.

For the control group, this lecture is a first introduction to the game and its environment; the experimental group attends the lecture after playing Cargo-Bot to contextualize direct instruction with the prior experience of game play.

4.3 The Pre-, Mid-, and Post-Tests

Each of our tests assesses the students' understanding of recursion in two ways: (1) Students trace a provided recursive function and determine its return value, and (2) stu-

dents write their own recursive functions to solve a given problem. The pre- and post-tests also contain several survey questions using a Likert scale to measure student engagement.

```
public int exec(int n){
  if(n == 0)
    return 0;
  else
    return n + exec(n - 1);
}
```

Figure 5: Code for the pre-test tracing question.

The pre-test includes a code-tracing problem (see Figure 5) in which students determine the value that is returned by the method call `exec(5)`. It also includes the task of writing a recursive function that determines whether a given string of nested parentheses is balanced, (i.e., every opening parenthesis has a matching closing parenthesis). The pre-test survey determines whether students have had any previous experience with Cargo-Bot by asking students if they have played it before. This item includes a number of distractors to reduce any possible confirmation bias that might occur if students suspect that Cargo-Bot may improve their understanding of recursion. The pre-test includes additional survey questions that ask the students to rate their knowledge of recursion (see Section 5.2).

```
public int tough(int x) {
  if(x < 0)
    return 2;
  else
    return x + tough(x - 1) + tough(x - 1);
}
```

Figure 6: Code for the mid-test tracing question.

The mid-test asks students to trace the code in Figure 6 and determine the value that is returned by the method call `tough(3)`. It also includes an iterative implementation of binary search and asks students to write a recursive version of the same search routine.

The post-test asks students to trace the code in Figure 7 to determine the value that is returned by `mystery(3, 35)`. It also asks students to implement the "bucket fill" tool—seen in typical image editing software packages—which fills all instances of a target color with a replacement color at a given location in the image. In addition, the post-test includes survey questions that measure the students' opinions about playing Cargo-Bot (see Section 5.2).

4.4 The Students

Our experiment uses two sections of AP Computer Science A at a public school magnet program in a large urban district, with one section serving as the control group and the other as the experimental group.

The control group contains 21 students; the experimental group contains 26. The control group is anecdotally the stronger of the two groups, as indicated by teacher commentary and an average first semester grade of 91.5% versus the experimental group's 87.7%. The first semester grades of

```
public int mystery(int a, int b) {
  if (b == 0)
    return 0;
  else if (b % 2 == 0)
    return mystery(a + a, b / 2);
  else
    return mystery(a + a, b / 2) + a;
}
```

Figure 7: Code for the post-test tracing question.

the control group are much less spread out than those of the experimental group, and the control group had only a single outlier of 78%, whereas the experimental group had three, each with a failing grade. In spite of the anecdotal evidence, however, there is no statistically significant difference between the groups' grades.

The control group meets from 9:50 a.m. to 11:30 a.m., before lunch, and the experimental group meets from 12:35 p.m. to 2:05 p.m., after lunch. Both classes follow the same schedule of topics, and neither group had any in-class exposure to recursion prior to our experiment.

5. RESULTS AND ANALYSIS

Using the pre-, mid-, and post-test results, we show with statistical significance that playing Cargo-Bot increases students' learning of recursion. In this section, we present the experimental results that justify this conclusion.

5.1 Test Results

We start by analyzing the writing portions of the pre-, mid-, and post-tests, in which students create their own recursive solutions. Figure 8 shows that students in the control group see a drop in performance from the pre-test to the mid-test, (i.e., after receiving direct instruction). After then playing Cargo-Bot, their scores increase by approximately 19.48%. By contrast, students in the experimental group, experience the greatest increase in performance between the pre- to mid-tests, which corresponds with their playing of Cargo-Bot. After the subsequent direct instruction, their test scores increase by just 4.48%.

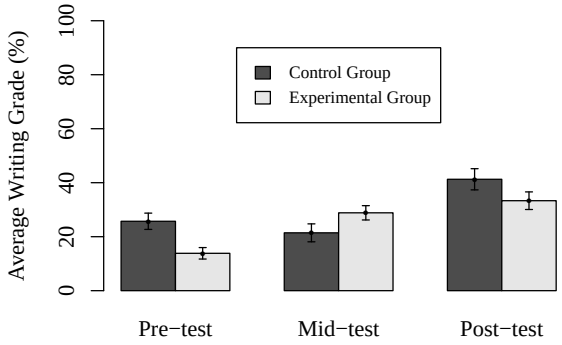


Figure 8: Average grade for the writing portion on each test.

Using a two-sample Student's *t*-test, we show that the learning gains (i.e., the difference in test scores) from the pre- to mid-tests for the experimental group are—with statistical significance—greater than those of the control group

Factors	Testing Group(s)	Mean Learning Gain (%)	t	df	p-value (2-tailed)
Pre- to mid-test writing scores	Control Group Experimental Group	-4.286 15.000	-2.0872	43.349	0.0428
Pre- to mid-test writing scores Mid- to post-test writing scores	Control Group	-4.286 19.841	-2.0664	36.245	0.0460

Table 2: Two-sample t-test shows that test performance improves significantly after playing Cargo-Bot ($p < 0.05$).

and that students experience greater learning gains in their abilities to write recursive functions after playing Cargo-Bot, rather than from direct instruction ($p < 0.05$, as seen in Table 2).

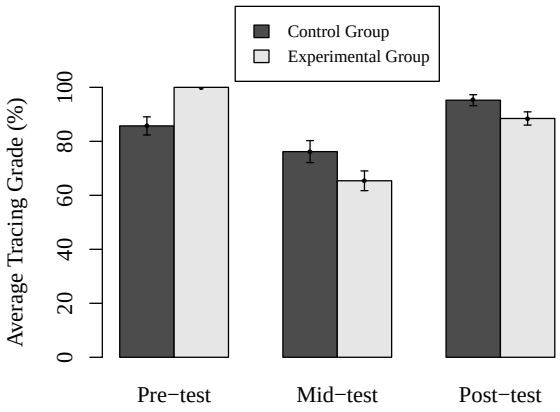


Figure 9: Average grade for the tracing portion on each test.

We observe very different results on the tracing portion of the pre-, mid-, and post-tests. As shown in Figure 9, students in both the control and experimental groups experience a decline in tracing performance from the pre- to mid-tests, then an increase from the mid- to post-tests. These results show that our new method of teaching recursion produces no significant difference in improving students' abilities to trace the execution of recursive functions.

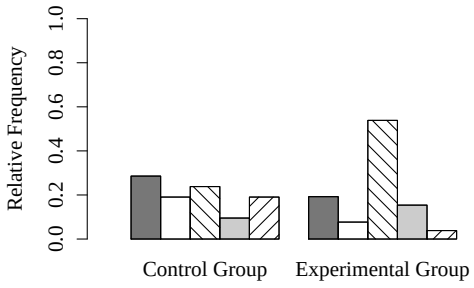
Thus, we see that playing Cargo-Bot significantly improves students' ability to write recursive functions, but it does not improve students' abilities to trace the execution of recursive functions. This is unsurprising, as Cargo-Bot does not explicitly involve code tracing; players instead use recursion at the conceptual problem-solving level, rather than as the procedural process described by Scholtz and Sanders [19].

Finally, we observe that there are various peculiarities where student grades do not monotonically increase from pre-test to mid-test to post-test. Our test questions were not validated, so particularly for the tracing questions, we may be seeing the effects of poorly chosen test questions.

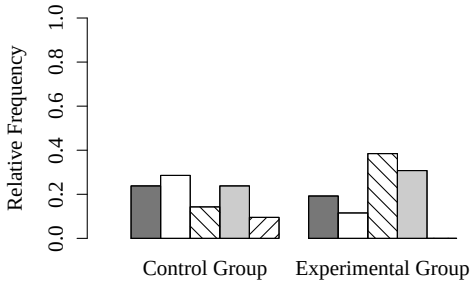
5.2 Survey Results

On the pre- and post-tests, we ask students a series of survey questions to gauge their own perceptions of their abilities to trace and write recursive functions, as well as their attitudes toward playing Cargo-Bot. In this section, we present, analyze, and compare the survey responses from students in the control and experimental groups.

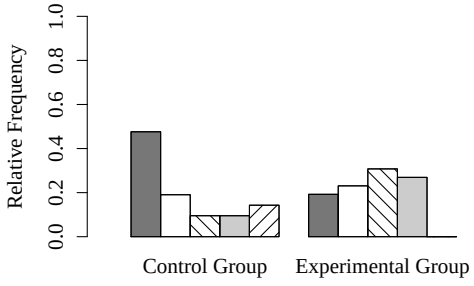
For both groups, over 90% of students have no prior knowledge of Cargo-Bot. This fact further strengthens our test results, because the vast majority of students experience significant learning gains after playing a game that they have never encountered before.



(a) "I understand recursion."



(b) "I can follow the execution of a recursive function."

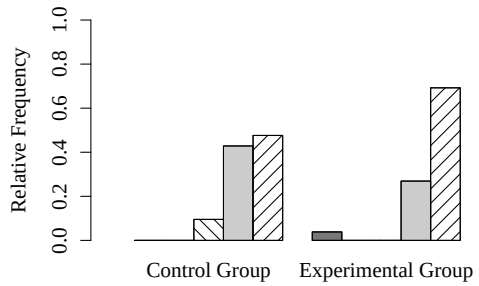


(c) "I can write a recursive function."

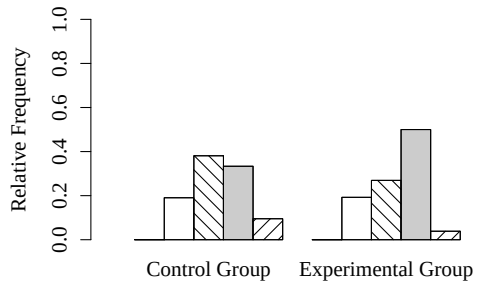
Figure 10: Pre-test survey responses.

Figure 10 shows the pre-test survey student confidence in their understanding of recursion. Clearly, it would have been instructive to have included these same questions on the post-test.

Figure 11 shows the post-test survey results. We see that the vast majority of students enjoy playing Cargo-Bot, and



(a) "I liked playing Cargo-Bot."



(b) "I am good at Cargo-Bot."

Figure 11: Post-test survey responses.

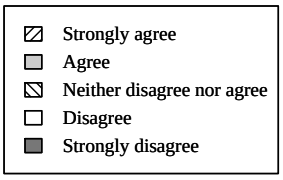


Figure 12: Survey response legend.

most are either neutral or confident in their ability to play Cargo-Bot. Finally, our third post-test survey question (not shown in Figure 11) reveals that 95% of the control group and 88% of the experimental group answered "Yes" to the question, "I realized that I used recursion in Cargo-Bot", which indicates that students are making an explicit connection between the game and recursion.

6. ANALYSIS

We originally posited that students find recursion difficult to grasp because they have no contextualized understanding of recursion. We then hypothesized that games such as Cargo-Bot could improve student mastery of recursion by providing this contextualized learning. To prevent embedding any implicit instructional bias within the game, we used an off-the-shelf game designed solely for entertainment, and we only modified this game to improve access and to instrument the results. As such, instruction for the experimental group was grounded in an authentically engaging environment.

Our results show that by playing Cargo-Bot, students do improve their ability to write recursive functions. The end gains for both groups indicate that Cargo-Bot improves learning regardless of the specific ordering of activities, and since Cargo-Bot does not include any tutorial or instructional component in the use of recursion, the improvement is likely due to the repeated practice in recursive thinking.

While we have not conclusively demonstrated that contextualization is the cause of this improvement, the larger gains for the experimental group suggest that grounding direct instruction in students' *prior* experience is preferred, a result that supports the social constructivist theories that learning is dependent upon the root context of the learner [15] and that instruction is best situated within the same context as its potential applications [13].

7. CONCLUSIONS

In this paper, we have proposed a new method of teaching recursion that combines direct instruction with Cargo-Bot, a game in which the user creates recursive programs to accomplish various goals. Our empirical results with 47 high school students in two separate classes have shown—with statistical significance—that our new teaching method improves students' understanding of recursion. Moreover, exit surveys show that nearly every student enjoyed the learning activity and that most were confident in their ability to accomplish the recursive exercises.

Given these promising early results, we invite the community to join us in expanding the scope of this initial study. We are repeating the experiment for two college courses, one an entry-level course for computer science majors, the other an introductory programming course for non-majors. These instances of the experiment will address some of the shortcomings of our previous experiment. For example, we will repeat the pre-test survey questions in the post-test.

We are also conducting a companion experiment with a college-level programming course, in which the control group learns solely through direct instruction with no mention of Cargo-Bot. Thus, we hope that this experiment will clearly demonstrate the benefit of our method of teaching recursion over a standard lecture-based method of teaching recursion.

As future work, we would like to modify our implementation of Cargo-Bot to illustrate the relationship between the program's execution and its stack-based flow of control, as we believe that this could improve students' abilities to trace recursive functions. More significantly, we hope to explore the use of other video games to teach other difficult-to-learn concepts, such as threading.

Acknowledgments.

We thank Alicia Beth, Mike Walfish, and George Veletsianos for their helpful comments on this research. We are grateful to Two Lives Left for their permission to re-implement Cargo-Bot in Javascript, and we thank Vicki Shan and Elynn Lee for their help in re-implementing Cargo-Bot.

References

[1] A. C. Benander and B. A. Benander. Student monks—teaching recursion in an IS or CS programming course using the Towers of Hanoi. *Journal of Information Systems Education*, 19(4):455–467, 2008.

[2] 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.

[3] D. I. Cordova and M. R. Lepper. Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization, and choice. *Journal of Educational Psychology*, 88(4):715–730, 1996.

[4] J. Davis-Dorsey. *The Role of Context Personalization and Problem Rewording in the Solving of Math Word Problems*. Memphis State University, 1989.

[5] J. Edgington. Teaching and viewing recursion as delegation. *Journal of Computing Sciences in Colleges*, 23 (1):241–246, Oct. 2007.

[6] G. Ford. A framework for teaching recursion. *SIGCSE Bulletin*, 14(2):32–39, June 1982.

[7] C. E. George. EROSI—visualising recursion and discovering new errors. *SIGCSE Bulletin*, 32(1):305–309, Mar. 2000.

[8] D. Ginat and E. Shifroni. Teaching recursion in a procedural environment—how much should we emphasize the computing model? *SIGCSE Bulletin*, 31(1):127–131, Mar. 1999.

[9] J. E. Greer. *An empirical comparison of techniques for teaching recursion in introductory computer sciences*. Ph.D. dissertation, The University of Texas at Austin, May 1987.

[10] K. Gunion, T. Milford, and U. Stege. Curing recursion aversion. *SIGCSE Bulletin*, 41(3):124–128, July 2009.

[11] W. Hsin. Teaching recursion using recursion graphs. *Journal of Computing Sciences in Colleges*, 23(4):217–222, Apr. 2008.

[12] R. L. Kruse. On teaching recursion. *SIGCSE Bulletin*, 14(1):92–96, Feb. 1982.

[13] J. Lave and E. Wenger. *Situated learning: Legitimate peripheral participation*. Cambridge University Press, 1991.

[14] D. Levy and T. Lapidot. Recursively speaking: analyzing students' discourse of recursive phenomena. *SIGCSE Bulletin*, 32(1):315–319, Mar. 2000.

[15] M. McMahon. Social constructivism and the world wide web—a paradigm for learning. In *ASCILITE conference. Perth, Australia*, 1997.

[16] P. L. Pirolli and J. R. Anderson. The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology*, 39(2): 240–272, June 1985.

[17] I. Polycarpou, A. Pasztor, and M. Adjouadi. A conceptual approach to teaching induction for computer science. *SIGCSE Bulletin*, 40(1):9–13, Mar. 2008.

[18] M. Rubio-Sánchez and I. Hernán-Losada. Exploring recursion with Fibonacci numbers. *SIGCSE Bulletin*, 39(3):359–359, June 2007.

[19] T. L. Scholtz and I. Sanders. Mental models of recursion: investigating students' understanding of recursion. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '10, pages 103–107, New York, NY, USA, 2010. ACM.

[20] R. Sooriamurthi. Problems in comprehending recursion and suggested solutions. *SIGCSE Bulletin*, 33(3):25–28, June 2001.

[21] J. Stasko, A. Badre, and C. Lewis. Do algorithm animations assist learning?: an empirical study and analysis. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 61–66, New York, NY, USA, 1993. ACM.

[22] M. A. Weiss. *Data Structure & Problem Solving Using Java*. Pearson Education, Inc., Boston, MA, third edition, 2006.

[23] S. Wiedenbeck. Learning recursion as a concept and as a programming technique. *SIGCSE Bulletin*, 20(1): 275–278, Feb. 1988.

[24] D. Wilcocks and I. Sanders. Animating recursion as an aid to instruction. *Computers & Education*, 23(3):221 – 226, 1994.

[25] D. L. Williams. The what, why, and how of contextual teaching in a mathematics classroom. *Mathematics Teacher*, 100(8):572–575, Apr. 2007.

[26] M. Wirth. Introducing recursion by parking cars. *SIGCSE Bulletin*, 40(4):52–55, Nov. 2008.