The Dissertation Committee for Joseph Tessler
certifies that this is the approved version of the following honors thesis:

Using Cargo-Bot to Provide Contextualized Learning of Recursion

Committee:

______________________________
Calvin Lin, Supervisor

______________________________
Michael Walfish

______________________________
George Veletsianos
Using Cargo-Bot to Provide Contextualized Learning of Recursion

by

Joseph Tessler

HONORS THESIS

Presented to the Faculty of the Computer Science Department of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

HONORS DEGREE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT AUSTIN

April 2013
Acknowledgments

I thank Calvin Lin and Bradley Beth for their valuable assistance writing and editing this thesis. I thank Alicia Beth, Mike Walfish, and George Veletsianos for their helpful comments on this research. Lastly, I thank my friends and family for waiting for me as I secluded myself to finish this work.

JOSEPH TESSLER

The University of Texas at Austin

April 2013
Using Cargo-Bot to Provide Contextualized Learning of Recursion

Joseph Tessler, B.S.
The University of Texas at Austin, 2013

Supervisor: Calvin Lin

This thesis presents a new method of teaching recursion in which students first play a video game that encourages recursive thinking. 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.
# Table of Contents

Acknowledgments iii

Abstract iv

Chapter 1. Introduction 1

Chapter 2. Background And Related Work 5

2.1 What is Recursion? 5

2.2 Other Instructional Approaches 7

Chapter 3. Experimental Design 9

3.1 The Video Game 11

3.2 Recursive Thinking in Cargo-Bot 14

3.3 Direct Instruction 14

3.4 The Pre-, Mid-, and Post-Tests 16

3.5 The Students 19

3.6 Experimental Limitations 20

Chapter 4. Results and Analysis 22
Chapter 1

Introduction

A student’s ability to understand recursion is an essential prerequisite for countless computer science algorithms. Without a proper understanding, students may dismiss recursion as hopelessly difficult and struggle with it throughout their studies [16]. Therefore, it is important that we provide introductory students with influential and effective recursion education.

There is nearly unanimous agreement that recursion is difficult to learn [15]. The concept of a function calling itself is simple, but recognizing the distinctions among different invocations of the same function, along with the amount of bookkeeping required for each recursive call, quickly complicates the scenario. Specifically, research shows that students struggle with the unfamiliarity of recursive activities [2]; the visualization of the program execution [11]; the backward flow of control after reaching the base case [21]; the comparison to loop structures [2]; and the lack of everyday analogies [18].

Teaching recursion is no less challenging and there exists an abun-
dance of literature that analyzes various instructional approaches [12]. These approaches include conceptual and abstract discussions of recursion and its control flow [7, 10, 19, 21, 22, 28]; comparisons to other topics and disciplines [8, 14, 20, 25]; and the use of visual aids and hands-on activities [3, 9, 12, 13, 23, 26]. Gunion et al. [12] claim that we need further “efforts to find useful and effective instructional approaches”.

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 [4, 5, 27]. In addition, our work continues the study of video games and the advantages of using them for educational purposes [1, 6, 17]. 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 command robots to accomplish certain tasks. In Cargo-Bot, players create simple visual programs to control a robotic arm so that it moves a set of crates to some specified configuration. Notably, Cargo-Bot does not provide explicit looping structures; recursion is the only means of iterating.

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

There are several benefits to this approach:

- Cargo-Bot contextualizes recursion in familiar terms of cranes and crates.
• As with most successful video games, Cargo-Bot is fun and addictive.

• Cargo-Bot provides multiple opportunities for students to practice thinking recursively, as they are given progressively more difficult problems to solve.

• The game encourages elegant uses of recursion by rewarding smaller Cargo-Bot programs with higher scores.

We propose a new approach to teaching recursion by allowing students to play engaging video games that essentially ask the user to create recursive programs to accomplish various goals. This thesis explores the question of whether these games could be used to improve the way that we teach recursion to introductory computer science students. We evaluate this idea by using an experiment conducted on 47 students across two AP Computer Science A high school classes, with one class serving as a control group and the other serving as the experimental group. We introduce recursion for the first time to students in both the control and experimental groups. The control group first listens to a traditional lecture on recursion, followed by a mid-test; then they simply play the video game before taking a post-test. By contrast, the experimental group plays the video game before learning any recursion; they then take the mid-test and have the same lecture on recursion before taking the post-test.

We find that the experimental group experienced a significantly greater increase in scores from playing the game than the control group did from direct instruction. Moreover, both the control and experimental groups experienced
the greatest increase in scores after playing the game.

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

Background And Related Work

In this chapter, we briefly define recursion and present common misconceptions that students develop when first introduced to the topic. We then analyze other proposed instruction approaches and place our work in the context of these works.

2.1 What is Recursion?

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, as shown in Figure 2.1 [24]. Every recursive function should obey the following rules:

1. The function must include at least one base case that can be computed without using recursion.
2. Every recursive call must progress toward a base case.

3. For efficiency reasons, each recursive call should avoid duplicating work.

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

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

(a) Print before recursive call.  (b) Print after recursive call.

Figure 2.1: Two functions that highlight the flow of control in recursion.

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 [11, 21]. For example, students may incorrectly conclude that the functions in Figures 2.1(a) and 2.1(b) produce the same output. However, with a proper understanding of the program stack, it is clear that \( f_1 \) will output \( x \) before the recursive call, which then outputs \( x - 1 \), so \( f_1 \) prints the numbers in descending order. By contrast, each invocation of \( f_2 \) is pushed on the stack before anything is printed; after reaching the base case, \( f_2 \) outputs 0, and then there is a backward flow of control through each invocation of \( f_2 \). So \( f_2 \) prints the numbers in ascending order.

Students are sometimes confused by simple but contrived examples of recursion, such as in Figure 2.1, that have better iterative counterparts, because the students then incorrectly associate recursion with loop structures [2]. Furthermore, such examples fail to show students the benefits of recursion [18].
2.2 Other Instructional Approaches

There is a large amount of existing research dedicated to improving the way that we teach recursion. In the following paragraphs, we present and discuss the various approaches proposed by these writers, then explain how our research builds on the previous work.

Ginat and Shifroni [10] 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 [22], 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 [7] suggested using the “concept of someone delegating a task to another person” as an example of a divide-and-conquer algorithm. Wirth [28] 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 [25] 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 [21] 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 under-
stand 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 [13] developed a recursive graph that “can help students understand the flow of a recursion process”. Similar to the work of George [9], Wilcocks and Sanders [26] use a program animator to assist students in “extrapolating a correct mental model of what recursion is”. However, experiments by Stasko et al. [23] found no significant result suggesting that algorithm animators assist 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. [3] 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.
Chapter 3

Experimental Design

The goal of our experiment was to determine whether Cargo-Bot could be used to improve the way that we teach recursion to novice computer science students. Specifically, we measured performance gains on a traditional code-based assessment over recursive routines before and after game play. We also measured the students’ learning gains achieved by playing the game both prior to direct instruction and after. Thus, our experiment tracked two groups of students. One group, the experimental group, played Cargo-Bot before receiving a lecture on recursion. The other group, the control group, received the lecture on recursion and then played Cargo-Bot.

To measure the effects of the various teaching components, we gave each group a series of tests, (1) a pre-test, which measured the students’ initial facility with recursion, (2) a mid-test, which evaluated the effectiveness of one teaching component, and (3) a post-test, which measured the effectiveness of the second teaching component. In particular, the schedules for the
two groups and their tests are shown in Table 3.1: The control group first received direct instruction covering recursion, followed by a mid-test; then they played Cargo-Bot before taking a post-test. Conversely, the experimental group played Cargo-Bot before formally learning recursion; they attended the recursion lecture after the mid-test, prior to taking the post-test.

Note that on day two the control group essentially becomes the experimental group, so it is perhaps a misnomer. Regardless, we refer to the control and experimental groups consistently throughout this thesis.

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

Cargo-Bot is a popular game for the Apple iPad in which users “teach a robot how to move crates”\(^1\), as shown in Figure 3.1. To teach the robot, the user writes a recursive program in a lightweight visual programming language. We rewrote the application in JavaScript—with permission from the original developer—to make the game accessible to all students with Internet access. We additionally modified the game to include user identification and tracking, allowing us to associate game play with individual students. The original game is not explicitly designed for educational use, and its developers are not directly involved in our research. Our rendition includes all of the 36 levels and GUI components that are found in the original video game.

\(^1\)http://twolivesleft.com/CargoBot
In Cargo-Bot, the player’s goal is to direct a robot to move a set of cargo blocks from some initial configuration (see the bottom of Figure 3.2a) to a specified goal configuration (see the top of Figure 3.2a). This direction comes in the form of a visual program that has simple commands, conditionals, and function calls for maneuvering the crane, as shown in Figure 3.2b.

For example, the user-defined program in Figure 3.2c consists of five
commands following the label F0: The first command tells the robot move
down, picking up or dropping any cargo that it might hold. The second com-
mand 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 holds no cargo. Finally, the last
command calls the function again to repeat the actions. Any function may
call itself recursively, and most levels require a recursive solution for higher
scores.

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 each level before moving on to the more challenging
ones. We removed this constraint, allowing students to explore more levels
within the allotted class time. Scoring is indicated by the assignment of 1
to 3 stars for a correct solution based on the number of register slots used.
The number of commands required for each star rating varies among scenarios
according to their difficulties.

Our version of Cargo-Bot is open source and freely available online\textsuperscript{2}.
However, there are other recursion-based video games available, such as Light-
Bot 2.0\textsuperscript{3}, that have similar traits and may be just as effective.

\textsuperscript{2}https://github.com/jtessler/cargo-not
\textsuperscript{3}http://armorgames.com/play/6061/light-bot-20
3.2 Recursive Thinking in Cargo-Bot

We now discuss the ways in which Cargo-Bot encourages students to think recursively.

Cargo-Bot levels are designed to be increasingly difficult as game play progresses. As a result, the recursive thinking required to complete a level, particularly with an optimal score, scales upward. The simplest examples use tail recursion—which is indistinguishable from iteration given the game’s visual programming environment. For example, Figure 3.2c shows a simple Cargo-Bot routine, $F_0$, which iterates by calling itself, $F_0$. In this limited example of Cargo-Bot recursion, it is not clear whether the program is calling a new instance of $F_0$ or simply executing a GOTO $F_0$ command—in effect, interpreting $F_0$ as a label rather than as a recursive function.

Later problems, however, require the user to make the nuanced distinction between the recursive and jump-to-label interpretations of the code. For example, Figure 3.3 utilizes the program stack to store the number of leftward movements needed to return to the leftmost platform. A non-recursive interpretation of the code fails to account for this implicitly stored value, and executes as at most one leftward movement.

3.3 Direct Instruction

This section describes the in-class lecture in more detail. Students in both the control and experimental groups received the same material for an
Figure 3.3: An advanced recursive solution requiring passive control flow.
The discussion began with an introduction to recursion similar to the one presented in Chapter 2. After defining recursion, we provided a basic overview of the program stack and worked through a series of tracing problems that exhibit the backward flow of control.

We concluded with a discussion of recursion within Cargo-Bot. We modeled recursion through examples that utilize the program stack, such as that depicted in Figure 3.3. Additionally, Cargo-Bot examples were 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 was a first introduction to the game; the experimental group attends the lecture after playing Cargo-Bot to contextualize direct instruction with the prior experience of game play.

### 3.4 The Pre-, Mid-, and Post-Tests

We now describe the three tests in some detail. Each test assessed the students' understanding of recursion in two ways. One required students to trace a provided recursive function and determine its return value; the other required students to write a recursive function from scratch to address a given problem statement. The pre- and post-tests also contained several survey questions using a Likert scale to measure student engagement.

The pre-test included a code-tracing problem (see Figure 3.4) in which students determine the value that is returned by the method call \texttt{exec(5)}. It
public int exec(int n){
    if(n == 0)
        return 0;
    else
        return n + exec(n - 1);
}

Figure 3.4: Code for the pre-test tracing question.

also included 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 determined whether
students had any previous experience with Cargo-Bot by asking students if
they had played it before. This item included 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
included the following survey questions:

• “I understand recursion.”

• “I can follow the execution of a recursive function.”

• “I can write a recursive function.”

The mid-test asked students to trace the code in Figure 3.5 and deter-
mine the value that is returned by the method call tough(3). It also included
an iterative implementation of binary search and asked students to write a
recursive version of the same search routine.

The post-test required students to trace the code in Figure 3.6 to de-
terminate the value that is returned by mystery(3, 35). It also asked students
public int tough(int x) {
    if (x < 0)
        return 2;
    else
        return x + tough(x - 1) + tough(x - 1);
}

Figure 3.5: Code for the mid-test tracing question.

to implement the “bucket fill” tool seen in a typical image editing software package, which fills all instances of a target color with a replacement color at a given location in the image. In addition, the post-test included the following survey questions:

- “I liked playing Cargo-Bot.”
- “I am good at Cargo-Bot.”
- “I realized that I used recursion in Cargo-Bot.”

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 3.6: Code for the post-test tracing question.
3.5 The Students

Our experiment included 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 contained 21 students; the experimental group contained 26. The control group was 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 box plot in Figure 3.7 shows how the grades in the control group are much less spread out than those of the experimental group. Moreover, the control group had only a single outlier of 78%, whereas the experimental group contained 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 met 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 followed the same time line of topics, and neither group had any in-class exposure to recursion prior to our experiment.

3.6 Experimental Limitations

This section describes possible sources of bias in our experiment. Primarily, we could not produce a truly controlled environment as we were using two preexisting classes. We had access to these classes because the author worked as a student teacher at the magnet high school while we conducted our research.

There were several limitations regarding the teaching environment. Given that the students were in a highly selective magnet school, these results might not be representative of a traditional high school environment. Moreover, the experimental group was more familiar with the author’s teaching style, because the author was the student teacher for their class. The author only taught students in the control group for the days of our experiment. However,
the author used the same direct instruction format that both classes had seen all year: a lecture with an accompanying slide deck.

The experimental timeline included other possible sources of bias. Two tests and direct instruction in a single day was a lot of drudgery and may have impacted the results of the control group’s mid-test and the experimental group’s post-test. The direct instruction for the control group was less relevant because they were also being introduced to Cargo-Bot for the first time. In addition, the experimental group had two nights to rest and digest the new information before taking the mid-test, unlike the control group.

We considered possible solutions to these limitations. For example, perhaps we should have included a third test group that only received direct instruction, providing a more concrete control group for unbiased comparison. However, as previously stated, we only had access to two classes and did the best we could with the limited resources available.
Chapter 4

Results and Analysis

Using the pre-, mid-, and post-test results, along with the survey responses, we show that playing Cargo-Bot statistically significantly increases students’ learning of recursion. In this chapter, we present and analyze the experimental results that justify this conclusion.

4.1 Test Results

In this section, we first analyze the writing portion results, then evaluate the tracing portion results. This allows us to determine whether playing Cargo-Bot helps improve students’ understanding of one question type over the other. As previously discussed, the pre-, mid-, and post-tests each included one question that required students to trace the execution of a recursive function and one that required them to write a recursive function from scratch.

As seen in Figure 4.1, students in the control group, who first were
Figure 4.1: Average grade for the writing portion on each test.

provided direct instruction, actually had an initial drop in performance on
the written portion of the assessments. Then, after playing Cargo-Bot, their
scores increased by approximately 19.48%. By contrast, students in the exper-
imental group, who first played Cargo-Bot, experienced the greatest increase
in performance between the pre- to mid-tests. Then, after direct instruction,
their test scores increased by just 4.48%.

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 experi-
Table 4.1: Two-sample $t$-test results indicate student performance on written test improved significantly after playing Cargo-Bot ($p < 0.05$).

mental group are statistically significantly greater than those of the control group 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 4.1).

In addition, we compared students’ learning gains on the writing portion from the pre- to mid-tests to the learning gains from the mid- to post-tests within the control and experimental groups. Students in the experimental group, who first played Cargo-Bot, then had direct instruction, experienced a monotonic increase in writing performance on each test. Conversely, students in the control group showed a drop in performance after the initial lecture before their performance improved after playing the game.

We observed very different results on the tracing portion of the pre-, mid-, and post-tests. As shown in Figure 4.2, students in both the control and experimental groups experienced a decline in tracing performance from the pre- to mid-tests, then an increase from the mid- to post-tests. In spite of this initial decrease, these results suggest that first playing Cargo-Bot or receiving direct instruction produces no significant difference in improving students’ abilities to trace the execution of a recursive function.
Thus, we show that playing Cargo-Bot significantly improves students’ performance writing recursive functions, but it does not seem to produce the same learning gains with students’ abilities to trace the execution of recursive functions. This is unsurprising, as Cargo-Bot does not explicitly involve code tracing; users focus on the use of recursion at the conceptual problem-solving level, rather than as the procedural process described by Scholtz and Sanders [21].
4.2 Survey Results

In this section, we present, analyze, and compare the survey responses from students in the control and experimental groups. On the pre- and post-tests, we asked 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.

![Survey Results Figure]

Figure 4.3: Survey results from the pre-test.

Over 90% of students in both the control and experimental groups...
played Cargo-Bot without prior knowledge of the game. This further strengthens our test results because the vast majority of students experience significant learning gains after playing a game they have never encountered before.

As shown in Figure 4.3, the majority of students in the experimental group respond with “Neither agree nor disagree” to each survey question in the pre-test. Perhaps students in the control group are more willing to share their self-assessment, evident by the relatively large proportion of “Strongly disagree” and “Strongly agree” in Figure 4.3. Given that both testing groups experienced the most learning gains after playing Cargo-Bot, this suggests that a student’s perception of his or her own understanding of recursion is not necessarily indicative of his or her performance.

![Survey results from the post-test.](image)

Figure 4.4: Survey results from the post-test.

After students in both the control and experimental groups completed both the game and direct instruction activities, there was nearly unanimous
agreement on the survey question, “I liked playing Cargo-Bot”. Moreover, there was a similar positive response to “I am good at Cargo-Bot”. In addition, more students in the control group realized Cargo-Bot is inherently recursive, but this to be expected, because they first attended a traditional lecture about recursion grounded in Cargo-Bot. Regardless, it is evident that the vast majority of students in both testing groups realized Cargo-Bot requires recursion and still enjoyed playing it. This shows that not only is this instructional approach effective at improving students’ understanding of recursion, but students overwhelmingly enjoyed the recursive activity and were confident in their ability to play the game.
Chapter 5

Conclusions

In this thesis, 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 has shown that our new teaching method improves students’ understanding of recursion, with statistically significant results. Moreover, surveys show that nearly every student enjoyed the learning activity and was confident in his ability to accomplish the recursive exercises.

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.
Bibliography


fifteenth annual conference on Innovation and technology in computer science education, ITiCSE '10, pages 103–107, New York, NY, USA, 2010. ACM.


