# Toward Using Games
# to
# Teach Fundamental Computer Science Concepts

---

A Dissertation

Presented to

the Faculty of Engineering and Computer Science

University of Denver

---

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

Jeffrey Michael Edgington

August 2010

Advisor:
Dr. Scott Leutenegger





ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Author: Jeffrey Michael Edgington
Title: Toward Using Games to Teach Fundamental Computer Science Concepts
Advisor: Dr. Scott Leutenegger
Degree Date: August 2010

# Abstract

Video and computer games have become an important area of study in the field of education. Games have been designed to teach mathematics, physics, raise social awareness, teach history and geography, and train soldiers in the military. Recent work has created computer games for teaching computer programming and understanding basic algorithms.

We present an investigation where computer games are used to teach two fundamental computer science concepts: boolean expressions and recursion. The games are intended to teach the concepts and not how to implement them in a programming language.

For this investigation, two computer games were created. One is designed to teach basic boolean expressions and operators and the other to teach fundamental concepts of recursion. We describe the design and implementation of both games.

We evaluate the effectiveness of these games using before and after surveys. The surveys were designed to ascertain basic understanding, attitudes and beliefs regarding the concepts. The boolean game was evaluated with local high school students and students in a college level introductory computer science course. The recursion game was evaluated with students in a college level introductory computer science course.

We present the analysis of the collected survey information for both games. This analysis shows a significant positive change in student attitude towards recursion and modest gains in student learning outcomes for both topics.

# Acknowledgements

I thank my advisor Scott Leutenegger for his many years of friendship and guidance. Without his patience and advice, I would never have finished this dissertation and degree. I also thank my committee members, Joel Cohen, Chris Gauthier-Dickey, and Roger Salters, for their time, advice, and comments. I have the deepest respect for each of the four of you.

I thank Ramki Thurimella for inviting me to consider graduate school. This invitation has led to many opportunities and allowed me to meet many wonderful and talented people. I will always be grateful.

For Mario Lopez, thank you for the many wonderful courses and sharing your knowledge and love of mathematics and algorithms. I also thank you for sharing many open problems with me over the years.

As for my graduate student colleagues, Mohammed Al-Bow, Dan Pittman, Gaurav Ghare, and Dan Daly, I thank each of you for the many conversations and meals over the years. Each of you made the difficult times bearable and the great times exceptional.

Finally and most importantly, I thank my wife, Angela.

Your love makes all things possible.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction and Overview

## 1.1 Introduction

There is considerable interest in using and designing video games as educational tools. This interest is nearly as old as video games themselves [119] and spans many different academic fields and disciplines including mathematics, geography, history, physics, politics, and language [9, 14, 17, 26, 27, 30, 41].

The Serious Games movement advocates the creation of "Games with a purpose beyond play" [35]. These games include those designed for training, education, health, and "advergames" (i.e. persuasive games with a political, social, or commercial purpose) [110, 163]. Two important parts of this movement are Games for Change [12] and Games for Health [13]. Games are being used to educate first-responders and by military organizations to recruit and train soldiers [2, 19, 39]. Numerous games have been created which draw attention to important social issues including world conflict, immigration, and world

hunger [8, 15, 18, 31, 36]. Video games have been created to educate the public about health issues and promote physical exercise [13].

Games are also being used to attract students to the study of computer science. This attraction is accomplished by offering students the opportunity to design and create video games [44, 45, 51, 61, 117, 161]. This curriculum has been introduced for several reasons: as an attempt to reverse declining enrollments in Computer Science, to provide the numbers of programmers which will be required by the entertainment industry and society in general, and because games are an emerging field of academic study. Recently, a number of computer games have been created for teaching specific computer programming concepts [49, 56, 64]. While this work is important, we believe that video games should be created which teach Computer Science concepts independent of any particular programming language. In this thesis we present results from investigating two video games: one for teaching boolean operators and expressions and one for teaching the concept of recursion.

## 1.2 Importance of Topic

### 1.2.1 Games

There is no question that video games are popular. In 2009, over 44 million video game consoles were purchased in the United States. This number includes only sales of the three most popular systems: Nintendo Wii, Microsoft XBox 360, and the Sony Playstation 3. Another 39 million hand-held game systems were also purchased during the same year. For these gaming systems, more than 350 million game were purchased in the same year [37].

We believe educational games should be designed and created in order to take advantage of this popularity.

### 1.2.2 Computer Science and Programming

The United States Bureau of Labor Statistics recently reported that there are just over 1.3 million workers with the occupational title of "computer software engineer" or "computer programmer". The bureau estimates the number of jobs in these fields will grow 21% by the year 2018 [4]. Games which teach programming and other Computer Science concepts could be an important supplement to a traditional Computer Science curriculum. Such games may also act as a recruiting tool and entice students to study Computer Science in school.

In 2006, it was estimated that 12 million people do some sort of "programming" at work. In the same year there were an estimated three million "professional programmers" [126]. Non-professional programmers are often called end-user programmers. Most end-user programmers have not received any formal computer science training and their main occupation is not programming. They write or modify programs in support of their main occupation. Since it is unlikely that these end-users will "go back to school" to study computer science, games for teaching computer science concepts to end-users would be a valuable learning tool.

### 1.2.3 Algorithm Visualization

Understanding data structures and algorithms is a critical part of an education in Computer Science. To further this understanding, many tools for visualizing the behavior of data

structures and algorithms have been created [20, 21, 139, 150, 153]. These tools allow the student to watch the data structure change during the execution of its defined operations. The student can usually control the execution by stepping backward and forward in execution time. Similarly, a student could watch an algorithm as it operates on a given set of data (i.e. watch the Quicksort algorithm as it chooses the pivot element, partitions the data to be sorted, and then recursively sorts each partition). An overview and history of these techniques can be found in [139, 150, 153].

A meta-study of algorithm visualization approaches has shown that visualizations are an effective teaching method especially if they require interaction and actively engage the student [100]. The same study also revealed that engagement is more important to the learning outcome than features provided by the visualization software or tool. Video games could provide an effective method for achieving student engagement. The design of educational games for Computer Science should be informed by what is known about effective algorithm visualization systems.

### 1.2.4 Boolean Expressions and Operations

Boolean expressions and operations are an integral part of conditional statements which are in every programming language and query language. Conditional statements are typically taught early in a first year programming course and understanding them is a critical part of learning to become a computer programmer. It is no surprise that conditional statements and boolean expressions are considered a fundamental topic of Computer Science [148] and Discrete Mathematics [46].

Unfortunately many beginning students have difficulty with this topic [86, 92, 96, 101, 129]. Part of this difficulty arises from the fact that there is often a mismatch between the name of a particular boolean operation and how that same word is used in everyday language. For example, "or" is often used to denote a mutually exclusive "choice" in everyday language (consider the question: Would you like tea or coffee? Most people will choose one or the other). In programming languages, "or", is not mutually exclusive (exclusive choice is typically represented by another operator called "exclusive or"). The operation named "and" is also confusing for similar reasons.

One goal of this thesis work is to aid students and non-professional programmers to more clearly understand these concepts. We propose and then study a specific video game designed to help achieve this goal.

### 1.2.5 Recursion

Recursion is an important topic in Computer Science and is considered one of the most difficult concepts to learn and teach [46, 114, 148]. Understanding recursion is critical for using some programming languages (e.g. LISP, Scheme, and the other functional languages). It is perhaps not as commonly used in other languages except for certain algorithms or situations. Recursion is often the method of choice when programming (or teaching) tree traversal algorithms for example.

There is a considerable body of research literature devoted to new methods of teaching this topic. Much of this literature describes new visualizations of the execution of a recursive function or new metaphors to help students understand how recursion works

[52, 53, 63, 71, 72, 89, 90, 95, 107, 154, 158, 47]. Some work has investigated the mental representations used by novices to understand recursion [85, 113, 147, 162].

A second goal of this thesis work is to create a video game which acts as a visualization of recursion. We propose a newly designed game and then examine its effectiveness as a teaching tool.

## 1.3 State of the Educational Game Field

When considering education and video games, there are two general areas of research. The first is understanding what players learn from playing video games created for entertainment. The second is creating video games to teach a particular topic or skill along with understanding how to create educational video games.

### 1.3.1 Learning from Entertainment Games

Research in this area aims to determine the learning effects, if any, of playing video games which are created primarily for entertainment. Relatively few studies have been conducted although there is a considerable amount of writing on this topic and learning from video games in general [43, 50, 76, 77, 138]. Studies have investigated both the physical and cognitive effects of playing video games.

The effect of video game play on specific physical skills has been studied quite thoroughly. These studies have shown improvements such as reduced reaction times, improved coordination, and improved visual attention [43, 87, 155].

There are also some results showing positive cognitive effects. According to [43], Cole [60] claims "long-term game playing has a positive effect on students' learning." Other studies, again according to [43], show improvements in "critical thinking and problem-solving skills" [144], "observation, trial, and error and hypothesis testing" [84, 87, 140], and "strategies of exploration." [138, 141]

### 1.3.2 Educational Games

The game, Oregon Trail, created in 1971 (later marketed by [123]) is perhaps the most well known educational video game. It was designed to teach the history and hardship of american settlers traveling across the west in the nineteenth century. Since then, a vast number of educational video games have been created. Among these are the Carmen Sandiego series, Math Blaster, Reader Rabbit, and The Magic School Bus series [23, 24, 32, 38]. These games teach geography, history, basic math, and reading skills. The two games, Rocky's Boots, and its sequel, Robot Odyssey [33, 34] were designed to teach fundamental logic design.

A virtual reality game called SMILE (Science and Math in an Immersive Learning Environment) was created in 2007 to teach science and math to young children (grades K through 5). Characters in the game can communicate with the player using speech, text, and American Sign Language. To succeed in the game, the player must use science, math, and language skills to create objects needed by the game characters. This requires players to learn not only basic science and math but language as well. The learning outcomes of this game have not been assessed [41].

Ke describes a study involving a series of eight games involving mathematical concepts [108]. The games included concepts "such as measurement, comparing whole numbers, solving simple equations, and mapping X and Y coordinates." Fifteen students played the eight games over ten sessions (each session was two hours). The results of the study show a significant attitude change toward mathematics but little or no change in mathematical knowledge.

Another recent game, Supercharged! [152], teaches the player about elementary physics (specifically concepts of electromagnetism). The player must pilot a spaceship through an "electromagnetic maze" by "placing charged particles" and altering an electromagnetic on the spaceship itself. This game was evaluated with nearly 100 middle school students. Students in the experimental group scored significantly better on a twelve question exam. Improvements were nearly identical regardless of gender.

### 1.3.3 Educational Games and Computer Science

There are three main categories of research: using games as motivation for studying computer science, using games as examples of computer science topics (e.g. data structures or design patterns) and designing games which teach computer science topics.

The creation and programming of video games are being used as motivation to study computer science [28, 111, 117, 136]. This is mainly an attempt to curtail the declining enrollments in computer science courses. Quite a number of game based courses and degrees have been recently created [48, 74, 97, 125, 164].

The programming of games is also being used to provide concrete examples of computer science topics. The annual ACM conference on computer science education (SIGCSE)

and other conferences publish papers on "nifty" examples or assignments. An example or assignment is considered "nifty" if it is fun or intriguing for students and easy to include in an appropriate course. Often the assignments are "open-ended" and therefor are scalable to a particular students abilities. Many of these assignments involve games [40, 131, 132, 133, 134, 135]. Game oriented programming projects have been created which involve object oriented programming topics (such as inheritance and polymorphism), data structures, and design patterns (such as visitor, concrete factory, state, and strategy). Most of this research reports that students enjoy this type of project because many students are highly motivated by creating video games [67, 93, 98, 99]. Few of these projects have been examined carefully to determine their effect on learning outcomes however.

Little work has been done in the area of creating educational video games which teach topics related to Computer Science. Most of this work has concentrated on games which teach computer programming in a specific programming language. The Game2Learn group, led by Dr. Tiffany Barnes, has created a number of this type of game [49]. Each game teaches a small number of topics or algorithms. For example, Wu's Castle [64] teaches arrays and loops by requiring the player to create arrays of snowmen using C++ syntax. Another game, Elemental: the Recurrence, teaches recursion and the depth first search algorithm using the C# language. These games have been shown to have a positive learning effect.

Lightbot [22] is a simple game which requires the player to assemble programming tiles to control a robot. It is necessary to program the robot to visit various "goal" squares and "light" them in order to solve the current game level and progress to the next. The programming language provided is simple: move forward, turn left, turn right, jump, and

"light" the current location. No educational studies have been conducted to determine if the game is an effective teaching tool.

## 1.4 Goals

We are interested in creating and testing the effectiveness of games which teach the concepts of boolean operators and recursion. We examine these two concepts because boolean operators and recursive thinking are fundamental concepts in the study of Computer Science [46]. Because we are interested in teaching concepts rather than implementation, game play does not involve any particular programming language.

We use these games as a method for concept visualization and engagement. We surmise that an educational video game should be effective for visualizing computer science topics if it can provide some of the necessary properties of effective algorithmic visualizations.

We agree with the simple design of Lightbot and with Fisch [70]: any programming or learning should be an integral part of the game. The games should also be simple so that game play does not overwhelm the fundamental concepts being taught.

## 1.5 Dissertation Outline

In the next chapter, we present a literature review of research directly concerned with games and education. We describe background and results concerning the learning effects of entertainment games, the Serious Games movement, and games designed for education in fields other than Computer Science. We then review the literature concerning games and Com-

puter Science: using games as educational motivation, programming games as examples of Computer Science topics, and finally games designed to teach computer programming.

Next, we provide another literature review of a more general nature. This review involves background theory of education, visualizations, and traditional techniques for teaching boolean operators and recursion. We start by defining and describing the concept of a mental model [105], its purpose, and how it relates to the concept of boolean operators and the concept of recursion. We also describe the currently known mental models that students may have concerning these two topics. We then describe what is currently known about algorithm visualization systems and their effectiveness in promoting positive learning outcomes. Finally, we present a review of published techniques and methods for teaching these two topics.

Next, we present the two case studies. The first is a game to teach boolean operators. The second is a game to teach the concepts of recursion. We describe the design and implementation for both games, data resulting from their evaluation, and analyze their effect on learning outcomes. We also discuss several interpretations of the results from these studies.

Finally, we conclude by presenting the contributions of this work to the fields of Computer Science and Education. We also discuss what went wrong and right, and possible future work in this area.

## 1.6 Summary

Video games are a relatively new educational media. Educational games have been designed to teach concepts from many areas: history, physics, and important social issues. Somewhat ironically, it is only recently that video games have been designed to teach computer programming. While we believe that this work is important (and it has been shown to be effective), additional video games should be designed to teach concepts of Computer Science rather than programming. Teaching concepts may serve two purposes: it could supplement a traditional Computer Science curriculum by providing additional concrete background material and it could be used to teach Computer Science concepts to non-majors and non-professional programmers without requiring more formal training. Games which teach concepts are also independent of any particular programming language implementation.

# Chapter 2

# Review of Literature

## 2.1 Introduction

Schneiderman [151] describes several advantages to using games as educational tools. First, "games provide a field of action that is simple to understand since it is an abstraction of reality – learning is by analogy." For computer science education games this "reality" is the set of abstractions created by software and algorithms. Second, games provide immediate graphical feedback to the player. The player can see the effect and results of their physical actions in direct fashion. This means that "there is no syntax to remember and therefore there are no syntax-error messages." Schneiderman also states that "error messages are unnecessary because the results of actions are obvious and can be reversed easily."

The graphical nature of video games is directly important since visual elements (images and icons) are often used as parts of explanations and mental representations of computer

systems. In a series of studies of computer users, participants were asked to explain various tasks and computer systems to another hypothetical user. These explanations were examined to determine whether they contained verbal elements, images, icons, production rules, or programs (one explanation could contain more than one of these styles). In one of the studies (with 607 secondary school students), 25% of the explanations contained images and 28% used icons in some fashion. This indicates that visual elements may be an important aspect of mental representations of computer systems and their behavior [159].

Many types of video games also support different categories of learning simultaneously. Gagne [75], as cited by Van Eck in [65], describes five categories of learning: "motor skills, attitudes, cognitive strategies, verbal information, and intellectual skills". Intellectual skills are further divided into "problem solving, rules, defined concepts, concrete concepts, and discriminations". These intellectual skills are hierarchical in nature (each depends on skills later in the list). For example, problem solving requires generating new rules from existing rules. Rules depend on defined concepts and concrete concepts. These concepts in turn depend on the ability to make discriminations which is the ability to determine or recognize differences between two objects, ideas, or concepts [65].

Van Eck [65] continues by describing a simple taxonomy of seven game types. These seven types are: action, role-playing, adventure, strategy (including puzzles), simulations, sports, and fighting games. Each type is then tied to a subset of Gagne's five sub-types of intellectual skills. This connection denotes that the particular type of game can support or facilitate the associated intellectual skill. Action games, for example, are tied to defined concepts and concrete concepts while most of the other game types are connected to the first four sub-types. Fighting games are the exception and do not generally include problem

solving. None of the game types are connected with the discriminations sub-type [65]. This means two things: fighting games should probably not be used to teach concepts which involve problem solving and it may be difficult to design video games which teach basic discriminations.

## 2.2 Games as Motivation

The creation and programming of video games are being used as motivation to study computer science. This approach is partly an attempt to curtail the declining enrollments in computer science courses and to attract under represented groups to study Computer Science. The use of game creation has several advantages: it can be taught using a variety of programming languages, environments, and programming philosophies, it is attractive to many students, and it does not inherently result in "watered down" courses [51, 111, 117].

Video games can be created in many different programming languages and environments. The literature contains reports of using ActionScript and Flash, C++, Greenfoot, Java, Python, and Scratch in game oriented courses [16, 28, 44, 45, 111, 117, 120, 137, 161]. Some environments (such as Greenfoot or Scratch) may provide better support for novice programmers however. This wide array of possibilities helps avoid one of the criticisms of the overall approach: that students will not be learning a "real" programming language. In any case, if students do start with one of the non-industrial languages, a transition to one of the mainstream languages can be provided [117].

The creation of video games is also not tied to any particular programming paradigm or philosophy. It can be used in object-oriented language courses ("objects early" or "objects late"), procedural language courses, or even functional language courses.

There is some evidence that the task of creating video games is attractive and motivating for many students including women. Cliburn reports that many students prefer game projects and assignments over non-game options (even when the projects become more difficult) [57]. In different study, a survey of 30 students (11 women) described in [117], the students responded favorably when asked if creating video games was "fun" (average response was 2.82 on a scale of 1 to 4) and whether they liked a game oriented course (average response was 3.26 on a scale of 1 to 4). The average responses for the 11 women were nearly identical: 2.9 and 3.18 for the two questions. Additionally, a game oriented approach appears to have a positive effect on student retention. Retention rates of 85% were also reported [117].

A series of studies described by Cliburn report similar but largely anecdotal results. In [59] more than 70% of students chose game oriented projects over non-game projects. This study interviewed six students and offers three recommendations: the assignment descriptions should provide as much structure as possible (particularly for beginning students), well-known games should be used in order to provide familiarity and to avoid confusion, and the games should have a graphical element [59, 58].

Becker [51] presents a case study which shows that the complexity and effort required to create computer games is substantially more than that required by non-game assignments. The computer games were an implementation of Solitaire, and the video game Centipede (both implemented using ASCII character "graphics") [5]. The non-game assignments

consisted of implementing a number of classes for geometric shapes and a simulation of a greenhouse. Students completed the games in the same amount of time as the students required to program the non-game assignments. This is an indication that student engagement and motivation may be as important (if not more important) than the course material presented. Students that are engaged and motivated are more willing to learn and do more work. It is difficult to believe that such courses are a "watered down" version of more traditional courses.

Using games in a course may not always be motivating to all students however. One study conducted by Rankin et. al. [142] presents some negative effects of using games. In this study, students "applied software engineering principles to the concept of game design" and created video games using the Gamemaker environment [11]. Twenty (self selected) students out of the 56 students in the course completed pre-surveys and post-surveys which measured attitude toward Computer Science courses, computer programming, and the pursuit of a degree in Computer Science. Twenty percent of the students responded that they were less likely to take another Computer Science course. Among the non-majors, 67% were less likely to take another Computer Science course. Attitudes toward computer programming increased for 25% of the participants and decreased for 40% of the participants. Interest in obtaining a Computer Science degree dropped for both non-majors (67% showed less interest) and majors (73% showed less interest). Student did report that "more time" allocated to the "conceptualization and implementation phases" of the project would have been beneficial because the project lasted only two and a half weeks [142]. These results were not compared to students in another non-game oriented course or to attitudes of stu-

dents from previous years which did not use games. This makes it more difficult to isolate the cause of the reported changes in student attitudes.

One study has shown that there are positive learning outcomes associated with this overall approach [117]. This study involved a rather small population of students however. The study was not an experiment but rather a quasi-experiment (i.e. there was no control or comparison group). The modest increases in programming knowledge that was reported is interesting but it is not known whether these increases are due to the introduction of games or some other factor (such as the instructor or the physical learning environment).

## 2.3 Programming Games as Examples

The programming of video games also provides concrete examples of computer science topics. Many of the "nifty" assignments use games for this purpose [40, 131, 132, 133, 135, 134]. Example games used are: Hunt the Wumpus, Rabbit Hunt, Minesweeper, Asteroids, Breakout, Missile Command, text-based adventure games, various card games, and Flip [3, 25, 6].

Huang proposes creating automated players for traditional board games [98]. Students create artificial players for classic strategy board games such as Connect-4, Mancala, Chinese Checkers, and Go. Creating a good artificial player for Go is very difficult and is in many ways an "open" research problem. The discussion regarding the game of Go is continued in [99].

Wallace and Margolis [160] use Sudoku puzzles and the board game, Clue [7], as part of an artificial intelligence course. They introduced competition between the student projects

by timing how quickly puzzle solutions were found. This was facilitated by a website which evaluated submissions and maintained a current "leader board". A number of students were highly motivated performing a literature search and implementing several advanced algorithms not discussed in class. In general, student response to these assignments and the competition was very positive but anecdotal in nature.

A different approach introduces the concept of "pre-games". In this approach, the students play a simple "game-like" version of an upcoming assignment "before designing and coding their own programs" [81, 82]. This allows the student to become familiar with the problem that they will soon have to solve. These simpler versions consist of playable algorithm animations. These playable animations allow the student to both control the algorithm animation and to play the game. The game play is not sophisticated but does include a fictitious story line (an archeologist named "Professor Raymond O. Folse" is searching an Aztec ruin for Montezuma's treasure room) which ties the elements of the assignment together. The "pre-games" presented algorithms involving stack (using Pez candy dispensers! [29]) and graph algorithms [81, 82].

Projects have been created which provide examples of specific algorithms and algorithm design strategies. Levitin et. al. present a series of puzzles whose solutions are examples of the algorithm design strategy known as divide and conquer. Puzzles presented include the Towers of Hanoi, tiling a region with triominos, finding a fake coin in a set of coins using a balance, and finding all anagrams of a specified word [118]. One example is the SIGCSE 2001 Maze demonstration program. This program provides a graphical interface and framework to allow students to create and study maze traversal algorithms [143]. Another example is a project which uses the "Incredible Rainbow Spitting Chicken". Stu-

dents create a chicken avatar which can move back and forth and can "spit" colored circles when a key is pressed. The circles float toward the top of the screen and are removed once they reach the top. Because there is no limit on the number of circles created this assignment can be used as an example for the implementation of linked lists [91].

Game oriented programming projects have also been created which involve object oriented programming topics (such as inheritance and polymorphism), data structures, and design patterns (such as visitor, concrete factory, state, and strategy). These ideas include implementing the card game Set, a clone of the video game, Every Extend, and the old unix game Rogue [68, 10].

The card game, Set [68], involves finding sets of three cards which match or do not match on their various features. To form a set the three cards must either all agree or all disagree on each of four features. The four types of features are: color (red, green, or purple), symbol (ovals, squiggles, or diamonds), number (one, two, or three symbols), and shading (symbols are either solid, open, or striped). Every possible combination of these features is represented giving a total of 81 cards in the deck. As an object oriented programming project [93], much of the implementation involves comparing cards to determine if they form a set and displaying individual cards. Card comparison is easily handled if the various features are implemented using the Flyweight design pattern. With this pattern, each feature value is created once and shared among all card which have that value (i.e. one object representing ovals is created and all of the cards with ovals reference that one object). This allows any comparisons to be based on references rather than values of objects (the author, Hansen, points out that in Java this allows the use of the regular comparison operators "==" and "!=" rather than having to implement an "equals" method for

each feature type). Determining if three cards form a set is then just a matter of checking if each feature is either equal across all cards or not equal across all cards. Displaying the cards is accomplished by using polymorphism along with the Strategy and Factory design patterns.

EEClone [80, 79], a clone of the video game Every Extend [10], is a two dimensional arcade style video game. The player moves an avatar trying to avoid a set of moving "obstacles". At the player's command an explosion is generated at the avatar's current location which will destroy the obstacles. Destroyed obstacles also explode starting a "chain reaction" of additional explosions. If enough obstacles are ultimately destroyed then the player is allowed to generate another explosion, otherwise the game is over. The player avatar is in one of four possible states: spawning, moving, exploding, or dead. These different states and the transitions between them are an appropriate use of the State design pattern. The Strategy design pattern is used to implement displaying each type of game element and any animation sequences. The Visitor design is used to handle collisions between game elements. The game is extensible in many ways: additional types of explosions and obstacles can be added along with new types of "power-ups" such as increasing avatar speed or temporary invulnerability.

Our own project idea, proposes the game of Rogue as a motivating example for the use of inheritance, polymorphism, and the concrete factory, state, and strategy design patterns [67]. Rogue is an "old school" Unix role playing game which originally used ASCII graphics. The player guides an avatar through a dungeon looking for the Amulet of Yendor. The dungeon contains various types of monsters, traps, and other items (such as armor, weapons, potions, scrolls, and wands). As a programming example, Rogue has two main

advantages. First, most students are generally familiar with computer based role playing games (e.g. World of Warcraft). This promotes student motivation and engagement. Second, the game provides a rich enough context to motivate the use of inheritance, polymorphism, and the design patterns listed earlier. For example, a concrete factory can be used to handle creation of all of the various monster objects. This makes the adding more monster types to the game very simple and clearly shows students the advantages of using the factory. The state pattern can be used to implement various avatar conditions such as paralysis, magical speed increases, and blindness (these game effects are temporary and eventually end). Strategy is used to model the different types of monster movement. For example, monsters can be "peaceful" and will continue to be until attacked by the player avatar. After being attacked the monster will usually become "aggressive". Both of these two types of movement can be implemented using the strategy pattern and reused for all of the different monster objects. Again this provides a concrete example for how this pattern is used. Use of these patterns definitely simplifies the overall implementation. One difficulty occurred with this project: even though the game was used in the third quarter of a CS1 course, student teams still struggled to implement the entire game (which requires thousands of lines of C++ code). Every team completed a playable (though limited) game however.

Most of the research discussed above reports that students enjoy these types of projects because many students are highly motivated by creating video games. Much of this evidence is anecdotal however [67, 93, 98, 99].

## 2.4 Learning Programming By Playing Games

A number of games including "Elemental: the Recurrence", Wu's Castle [64], The Catacombs, Saving Sera [49] (among other games) from the Game2Learn group [56] are designed to teach the player various computer programming concepts. These games teach simple programming involving arrays, loops, recursion, and tree traversal.

Wu's Castle [64] teaches the concepts of arrays and loops. This game consists of two sub-games. The first involves array manipulation. In this sub-game, the player creates arrays of different types of "snowmen". The player is first presented with for-loop parameters (using C++ syntax) which they are allowed to change. Then the player chooses the loop body from a menu of choices. A game character then acts out the loop execution in a visual manner. A series of "missions" are presented to the player. Each involves creating certain types of snowmen in the cells of either a one-dimensional or two-dimensional array. Once the player successfully completes one "mission" they are allowed to proceed to the next. In the second sub-game, the player avatar can interactively walk through a game level which represents the execution path of a loop. As the avatar moves, the game reports which part of the loop's code is being "executed". The authors report that playing the game before solving homework programs helped students "create a deeper, more robust understanding of computing concepts." The gains were even greater for more difficult problems [64].

"Elemental: the Recurrence" is a game designed to teach the concept of recursion. The player is asked to write several programs in C# to recursively guide an avatar. The avatar collects tokens (which represent portions of the avatar's sanity) by visiting nodes in a tree structure. To collect all of tokens, the avatar must successfully perform a depth first traversal of a tree structure. A study of the learning effectiveness was conducted.

Participants in the study completed a pre-survey, played the game for 40 minutes, and then completed a post-survey. The surveys contained five questions directly concerning recursion (the questions were not published). "Elemental: the Recurrence" wraps a simple game around typical homework assignments. This provides effective motivation (beyond receiving course grades) for the player to complete the assignments. The game was shown to be effective in improving learning outcomes [56].

All of these games from the Game2Learn group suffer from the fact that the programming tasks are "separated" from game play. The concepts to be learned are not an inherent part of the game world. This forces the player to "switch roles" between playing and programming during game play [146]. This is related to the idea of "gratuitous incidents" described by Laurel [115]. These are "incidents which have no direct bearing on the plot" or outcome of the story. These kinds of incidents are often exhibited in an effort to "provide intrinsic motivation in educational programs by interspersing problem-solving or tutorial segments with pieces of games." [115].

These are not "fatal" flaws however. Each of the games has been shown to be effective in improving learning outcomes. Our major criticism of this approach concerns game play (In role-playing games, sorcerers do not code in C++). If the goal of the video game is to teach computer programming in a more or less direct fashion then new programming languages should be designed which fit and match the world design of the game. Another minor criticism is that these games are tied to a specific programming language. It may be difficult to modify the games for use with another language.

In the game Lightbot (produced by Armor Games), the player is presented with a series of puzzles [22]. The puzzles require the player to write simple programs using a graphical

language to make a robot move and "light up" designated goal tiles in a simple world. The language is simple and has a small number of commands: move forward, turn left, turn right, jump, and light the current square. There are no mechanisms for iteration or conditional statements. Each command is represented by a tile (with an appropriate graphical symbol) and programs are described by creating sequences of these tiles. The program can also include two user-created functions and there are command tiles which allow the invocation of these functions. The functions do not allow parameters (there are no variables in the language). The puzzles are challenging because the user-created functions and the overall program cannot consist of more than 28 instructions total (12 in the main function and 8 each in two other functions). This limitation is particularly important because there is no construct for performing iteration. Because it is not possible to pass function parameters and there are no conditional statements, it is not possible for Lightbot programs to contain recursive functions.

Since the goal of Lightbot is to control the robot using a program the player does not have to "step out of the game" while playing thereby avoiding the "magic circle" problem [146]. Creating the program is an integral part of the game itself. This is the result of good game design. The simplicity of the design along with the immediate graphical feedback of the program effect create considerable player engagement and interest.

The sequence of puzzles is also well-designed. First a number of tutorial puzzles are presented which teach the player the purpose of each command tile and how to create programs. After these basic puzzles the difficulty increases steadily. Each new puzzle requires more creativity and careful thought than the earlier puzzles. This further increases

the level of player engagement. It is this high level of engagement and simple design which make Lightbot interesting pedagogically and worth further study.

The sequence of Lightbot puzzles probably improves understanding of command sequences and some aspects of functions (particularly code reuse and invocation). The game cannot improve understanding of variables, iteration, or conditionals since the programming language does not include these features. We are not aware of any educational studies involving this game.

## 2.5 Summary

The study of video games is an important emerging field of academic study. Video games are interesting both because of their potential educational impact and as an art and media form. Video games are being used to attract students to study Computer Science and as a method to retain them once they start. Games are believed to motivate and engage students when they are used as course projects and assignments. There is some evidence that this is, in fact, the case for many students. The motivation and engagement arise from the interactive and graphical nature of the video games themselves though competition may also play a role for some students [116].

Our two video games can be considered a continuation of the "pre-game" work of Giguette [81, 82]. He proposed playable animations as an aid toward student understanding of upcoming implementation assignments. These playable animations generally contain two sets of controls: one set for the algorithm visualization and one set for game play. Our

thesis work aims to tie game play and the educational content in a more seamless fashion while still acting as an effective visualization.

Our video games parallel the work of the Game2Learn project [56]. Their important work shows that video games can be created which improve student understanding of the programming of various algorithms. Our work creates games which try to teach underlying and fundamental concepts rather than programming in any particular programming language.

# Chapter 3

# Background Theory

## 3.1 Mental Models

Veer [159] defines a mental model, "... as any set of mental representations that is used by a human being to understand a system." In general, mental models represent some portion of the structure of a system in the same manner that "an architect's model or a molecular biologist's model" represent the structure of a building or a molecule. Johnson-Laird continues: "Like these physical models, a mental model is also partial because it represents only certain aspects of the situation. There is accordingly a many-to-one mapping from possibilities in the world to their mental model" [105]. In other words, one mental model represents many similar systems or situations. New mental models may be created from visual perception, combining existing mental models, or comprehension of written or verbal discourse [104].

A mental model is viable if the model allows the person to successfully apply it to the concept or situation which is modeled (otherwise it is called a non-viable model). Note that the model does not itself have to be correct; it is the successful application which makes the model viable. Non-viable models may lead to misconceptions about the modeled system or situation. Because we are interested in creating games to teach boolean operators and recursion, we are specifically interested in different mental models students may have of these concepts.

### 3.1.1 Models of Boolean Expressions

One traditional method for teaching boolean expressions starts by mathematically defining the three common operators (which we will denote using AND, OR, NOT to avoid confusion). These operators are shown as functions which act on a set of variables and that have a value of "true" or "false". These functions are usually visualized using truth tables. The information presented is certainly accurate. If there is a drawback with this method it is that some students have little or no experience with formal mathematical definitions. These students may have difficulty constructing appropriate mental models since they do not have much context to build upon.

Another drawback is the choice of names for the operators themselves. The natural language meaning of the word "and" in particular is largely dependent on conversational context. The boolean operator AND has no such problem of ambiguity or dependency on context. This may cause confusion when the boolean operator is seen to "behave" differently than what the student knows from their conversational experience. This was

shown in a study by Pane [130]. Because of this confusion has been proposed that novice programming languages should not use the word “and” to represent AND [121].

Novice programmers often have difficulties creating correct boolean expressions and comprehending existing expressions [83, 86, 96]. Much of this difficulty is due to fact that the words used to signify the boolean operators often have different meaning when used in everyday conversation. While the operators in the programming language have well defined semantics, the corresponding words in english are ambiguous and the meaning often depends on the context. For example, suppose someone is asked to count the number of blue and green symbols in Figure 3.1. The person may respond that there are no symbols which are both blue and green (so the answer is zero) or they may interpret “and” as a disjunction and report an answer of six (the sum of the number of blue and the number of green symbols).

Most of the systems used in Information Retrieval use queries based on boolean operators. Users specify a query using keywords and common boolean operators (such as NOT, AND, and OR). Because several studies have shown that users have difficulty accurately converting a natural language description into a correct boolean query some systems attempt to perform this translation automatically. There has also been work to find better methods for representing boolean queries [86, 124, 130, 151].

In mainstream computer programming languages, a boolean data type is often provided and textual symbols or keywords are provided for commonly used boolean operators. Variables which have this type can have only two values “true” or “false”. In languages where a data type is not provided, the value of zero is often considered to be “false” and any other value is considered to be “true”.

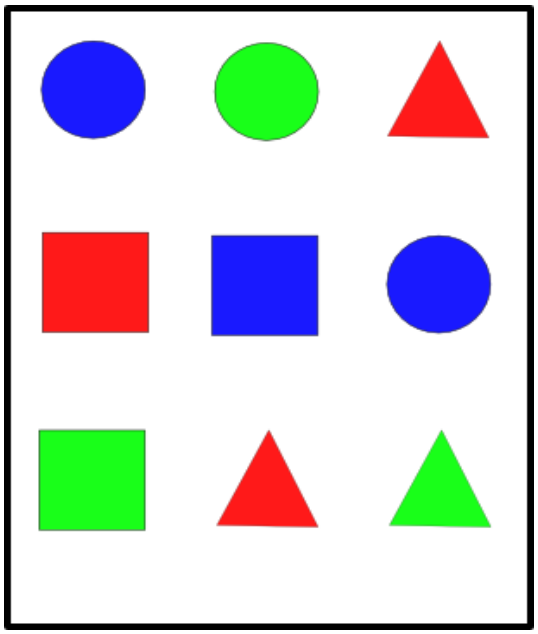

**Figure 3.1:** *Array of Various Shapes*

Another source of difficulty may be caused by mental models themselves. Johnson-Laird has shown that "... mental models represent only what is true, not what is false. This principle of truth applies both to premises as a whole and to clauses within them. For premises as a whole, models represent only the possibilities that are true" [105]. Even if we create viable mental models of boolean expressions these models may not directly assist us in determining what makes an expression false.

Creating viable mental models of boolean expressions may be difficult due to lack of experience and confusion arising from previous natural language knowledge. We propose to use a video game to increase familiarity with boolean operations and to lessen the effect of the ambiguity arising from experience with natural language. We hope that after playing our game, it will be easier for students to create viable mental models of boolean operations.

### 3.1.2 Models of Recursion

We are also interested in mental models of recursion. A mental model of recursion is viable if it allows one to "accurately and consistently represent the mechanics of recursion" otherwise the model is considered non-viable. Non-viable mental models arise from not understanding "the mechanisms of recursion or ... concepts fundamental to recursion" [85]. A viable mental model concerns the functions of planning and execution as described above.

Kessler believes that part of the difficulty with learning recursion is that "... there is no natural everday activity than can serve as a precursor for recursion" [109]. It may be difficult to find recursive activities but there are real world examples and situations that imitate recursion however. Russian nesting dolls, pictures such as the packaging for Droste Cacao, and many of M. C. Escher's paintings, are all examples of self-referential or recursive artwork.

In a study of first year students, Gotschi et. al. observed one viable model (the copies model) and a number of non-viable models (looping, active, step, magic, return-value and other odd models). [85]. It is interesting to note that this study asked participants to understand and write programs in Scheme (a language which relies heavily on recursion).

The control flow of execution of a recursive function can be divided into two parts. The active flow consists of times when the control is being passed to new function invocations. Passive flow consists of those times when the control is being returned back from those invocations [78]. Both of these flows must be present in a viable mental model of recursion.

The Copies Model is described by Kahney in [106]. In this model, a new copy of the recursive function (along with a new environment) is created each time the function

is invoked. The new copy is executed and any results are returned to the calling function. The copies are created during the active flow of the recursive process. During the passive flow, these copies and the associated environment are destroyed as they complete their computation. This model can be viewed as a special form of delegation as described in [66]. This model is always viable.

Expert programmers are believed to possess this cognitive model. This belief is mostly based on a survey conducted by Kahney [106]. In the survey, participants were presented with a simple programming problem. The problem consisted of performing the transitive closure of a property on a simple set of data. Participants were presented with three possible "solutions". The first "solution" was actually incorrect and if chosen would demonstrate that the participant had a non-viable cognitive model of recursion. The second solution is tail-recursive and does not require any calculation in the passive flow. The last solution is the most complex with the recursive call occurring in the middle of the function and requiring calculation in during the passive flow. The study assumes that both of the latter solutions should be chosen if the participant possesses the Copies cognitive model of recursion. After choosing the solutions that were believed to be correct, the participants were asked to describe why each solution was chosen or rejected. Eight out of nine expert participants chose both correct recursive solutions [106].

Kahney also included "novices" in the study. Unfortunately previous programming experience was not included in the survey questions so it was unknown whether the participants were truly "novices" or not. Only one participant could be strongly classified as possessing the Copies model. Four participants showed strong evidence for possessing

the Looping model (described below). These results led to an early attempt to describe different cognitive models of recursion.

The looping model occurs when, "... the recursive procedure is viewed as a single object rather than a series of instantiations and thus recursion is seen as a form of iteration. The solution is calculated once the base case is reached, thus the base case is seen as the stopping condition of the loop." [85]. Because the recursion is viewed as iteration, neither the active flow nor the passive flow are recognized (this is because there is no recognition that the procedure is being invoked more than once). The looping model is viable for recursive functions where the final result can be calculated at the point where execution reaches a base case [147].

Another model, the Active Model, does not account for the passive flow only the active flow. The recursion is assumed to end when the execution reaches a base case. Any calculations performed during the passive flow are ignored [85]. The difference between this model and the Looping Model is that the active flow is correctly modeled. The Active Model is similar to the Looping Model in that it may be viable for some recursive functions but is not viable in general.

In the Step Model, only the conditional structure of the recursion is evaluated. There is no apparent recognition of either the active or passive flow. The student has "no understanding of recursion". The incorrect evaluation "... involves either execution of the recursive condition once, or of the recursive condition once and of the base case." [147].

Sanders also discusses the Magic or Syntactic Model. In this case, the student shows no clear understanding of how recursion works but the student is able to match on syntactic elements. This is often a precursor to the Copies Model [147].

Sanders et. al. report that students should be presented with a recursive function which involves both the active and passive flows in the evaluation of the result earlier rather than later. In other words, it may help to show a complex example sooner rather than starting with simple examples and progressing toward more complex examples. [147].

One of our video games acts as a visualization and example of recursion. During game play, conditions can occur where the current game level is suspended and a new game level is created. All of the unfinished game levels are shown superimposed on one another as a stack of game levels. We believe this representation will aid the formation of the viable Copies mental model.

## 3.2 Visualizations and Representations

A stated earlier, the meta-study by Hundhausen et. al. [100], has shown that visualizations can be an effective teaching tool. According to Hundhausen, in order to be effective, the visualization software must engage the learner. The level of engagement is more important than features of the visualization software.

An engagement taxonomy was created by a 2002 ITiCSE working group and is described in [127]. The taxonomy includes six levels of engagement: No Viewing, Viewing, Responding, Changing, Constructing, and Presenting. The first level is to not use the visualization and the second level consists of passive observation. The "Responding" level requires the student to view the visualization and answer questions asked by the visualization system (or some other source). The next two levels require the student to alter an existing visualization or create a new one. Alteration usually consists of specifying new

parameters or input to the algorithm being viewed. The final level consists of having the student present a visualization to an audience [88].

The meta-study [100] presented a number of interesting results. Nine of the examined studies engaged the participants at the passive or "Viewing" level. Only three of these studies showed significant results. For the twelve studies which engaged the participants at higher levels, ten showed significant improvements in learning outcome.

Two studies used the "Responding" level and their results are somewhat contradictory (one study [54] reported a positive learning outcome and the other [103] reported no effect). A later study [88] reports a significant positive effect. There were also two studies involving the "Changing" level. Both studies show improvement over engaging students at the passive level. It is currently unclear whether construction of a visualization is significantly better than the previous levels and as of 2003 no studies have been conducted concerning the "Presenting" level [88].

In educational games, the concepts being taught are often portrayed graphically [51]. It is unclear where games fit into the engagement taxonomy described above however. Clearly video games require engagement at least at the "Viewing" level but is unclear whether playing and reacting to a game is equivalent to answering questions (i.e. participating at the "Responding" level).

A more recent work [122] lists a "significant subset of features that an increasing body of educational research has found necessary if visualizations are to result in actual learning on the part of the viewer." The list includes: accompanying text, variety, interactive questioning, "the ability to move backward and forward through the visualization", and "the ease of integration into course management data".

Accompanying text consists of additional information (i.e. textual descriptions of the algorithm being viewed or a view of program code which matches the current state of the visualization). This additional information is "necessary to provide the learner with an explanation of the algorithmic context for the graphics they are watching." (originally from [145] and cited in [122]).

Variety is the ability of the student to alter the "parameters" of the visualization. This allows the student to examine the effect of various input on the behavior of the algorithm and its visualization. This greatly increases the power of the visualization [122, 127].

The student should also occasionally stop and reflect on what the visualization is showing them. This is accomplished by asking the student questions during the execution of the visualization. It is important that the questions avoid repetition and predictability [103, 122].

Most visualization systems provide the student with the ability to step both forward and backward through a visualization. This feature is provided so that if the student "becomes confused" they can easily repeat portions of the algorithm animation [122]. Finally, student responses should be recorded in order to provide additional feedback and to assess the students' progress in understanding the algorithm being visualized [102, 122, 145]

Most of these principles are relevant to the design of educational games for visualizing computer science topics. For many games, additional text is provided as game manuals, in-game instructions and tutorials. Variety, in many games, is "built in". The player of the game provides the input and the game responds with the resulting behavior. The necessity of reflecting on what is happening during game play is also "built in" to well designed and "deep" games [51]. The ability to move backward and forward through time is available in

games (usually in a limited fashion). Most games do not provide the explicit ability to step backward and forward in game play. If something "goes wrong" and the player becomes "confused" then the player must retry some portion of the game (either continuing with the current "life" or by restarting with the next "life" or a new game). The final property can be accomplished by having the educational game record relevant game play for later examination.

Our video games follow several of these principles. Regarding the principle of additional text, the two games are very simple requiring minimal instructions and advance from simple situations to those which are more complex. Concerning the principle of variety, while we do not explicitly allow the person playing to provide "parameters" in order to explore their effects, the games do show quite a number of different situations. Our games do not provide the ability to step backward during game play. The player can only restart the game from the beginning. The games also do not record game play.

### 3.2.1 Representations of Boolean Expressions

Boolean expressions can be represented visually using many different methods including truth tables, Venn diagrams, and decision tables [86, 124].

Another method is the Shneiderman's Filter-Flow model [151]. In this method, criteria selection boxes show a list of possible values for each particular attribute in the expression. Criteria selection boxes are connected together by attaching the output of one box to the input of other boxes. Width of the "flow" denotes the size of the selected set. An AND operator is shown as a series of these criteria selection boxes. An OR operator is shown as either multiple selections within a single criteria selection box (when the OR in-

volves one attribute) or as parallel criteria selection boxes (when the OR involves multiple attributes). The NOT operator is represented by inverting the currently chosen values in a criteria selection box.

Pane [130] proposes another representation called match forms. A match form is a tabular representation of a boolean expression. Each entry in the table corresponds to a property described in the expression. For example a match form concerning geometric shapes might have three table entries, one for the shapes name, one for the shapes color, and one for the size of the shape. Each entry in the table is a disjunction of the criteria desired (i.e. the entry for color might be "red or blue"). The overall boolean expression is the conjunction of each of the table entries. Pane showed that this representation is often easily understood by novice programmers [128, 130].

### 3.2.2 Representations of Recursion

There have been many attempts to develop effective techniques for teaching recursion. These techniques include visualizations, new metaphors and analogies, and methods for understanding student mental models and misconceptions. According to Ben-Ari [52], there are two general approaches: using analogy with real-world objects and using diagrams of activation records. The analogies demonstrate that recursion occurs "naturally" in the real world. Diagrams of activation records show how recursion is usually executed on a computer.

Examples described in [52] include russian nesting dolls or puzzles. Russian nesting dolls are sequence of different sized hollow figurines, each containing the next smaller sized doll inside it. The dolls are a simple model of recursion. Each doll represents one

instance of the recursive problem (or one invocation of the recursive function). Since each doll is a smaller replica (or clone) of the first and largest doll, this analogy should directly aid the formation of a viable cognitive model of recursion. The sequence of real dolls is typically linear and cannot directly represent a function with more than one recursive calls.

One puzzle, the Towers of Hanoi, is often used in teaching recursion. The puzzle is to move a stack of disks (each of different size) from one of three pegs to another with the restriction that no disk can be placed on top of a smaller disk. Moving a stack of n disks to a goal peg can be accomplished by moving a stack of $n-1$ disks to a different peg than the goal, followed by moving the last disk to the goal peg, and then moving the stack of $n-1$ disks to the goal peg on top of the last disk. Moving the stack of $n-1$ disks can be accomplished recursively in similar fashion. This is an interesting real-world example of recursion because it involves two recursive invocations (to move the $n-1$ disks before and after moving the last disk). While the puzzle can be solved recursively, it is unclear how the Towers of Hanoi models recursion in general.

Diagrams are used to visualize the current state of the recursive procedure with the stack of activation records and including all variables and their values. Ben-Ari criticizes both these approaches claiming that the commonly used analogies are "weak" and there is little connection between the real-world example and any recursive computer algorithm. The second approach is problematic because one must explain the stack data structure and activation records for which the student may not be ready [52].

There are many different diagramming techniques for designing and specifying computer software. Several of these techniques can be used to form the basis of visualizations of recursive procedures.

Data Flow Diagrams (DFD) show the flow of data through the procedure. Data Flow Diagrams do not show order in which event occur however [156]. The representation of a recursive function consists of one "process" which consumes the initial input, may send output back to itself, and ultimately produces its final output (this is a graphical representation of the "looping mental model" misconception). If the DFD is modified so that it shows each recursive invocation as a separate "process" then the diagram more clearly represents the recursion. This is a graphical representation of the Little People Metaphor where the Little People can be "cloned". We describe the Little People Metaphor in Section 3.2.7.

Nassi-Shneiderman diagrams [156] show the flow of control of a procedure. A recursive call is depicted in the same way as any other function call. The diagram does not show that a recursive call is to a new copy of the function. There is nothing explicit in the diagram which assists the learner in acquiring a correct mental model. If the recursive function call is replaced (graphically) by a smaller copy of the entire function diagram then the correct mental model becomes explicit. The complete diagram then becomes a depiction of the execution stack (one of the main visualization methods of recursion).

### 3.2.3 Classic Teaching Approaches

One standard approach is to describe in detail when to use recursion to solve a problem. This according to Ford [71] "also gives some indication of how it is used." Specifically "Recursion may be used to solve a problem if that problem exhibits properties such that:

- Solving any instance of the problem (except the smallest) requires solving one or more smaller instances of the same problem.

- The smallest instances of the problem can be solved directly, usually in a trivial manner.

Another approach is to teach recursion as an abstraction using the concepts of procedure invocation and mathematical induction [72].

These two traditional approaches consist mainly of presenting the student with definitions of what it means to solve a problem recursively and recursion itself. Both approaches (especially the second) are mathematical in nature. Since few students have a strong mathematical background these approaches do not always provide much context or build on what the student currently understands. This inadequate background makes it more difficult for the student to construct a personal mental model of recursion.

Another method is to teach recursion by presenting the students with a series of ”classic” problems and their recursive solutions. These classic problems usually include the following: Factorial, Power, Binary Search, Fibonacci, and Towers of Hanoi. Other less used problems include: the Eight Queens Problem, Palindromes, and the Quicksort algorithm [107]. This approach may assist in the construction of a viable cognitive model of recursion. In fact, later research has shown that the complexity of the problems used do matter. If the problem is too simple then the student does not understand the role of both the active and passive flows in the function execution [147].

Bruce et. al. report on their experience when students are taught structural recursion before iteration [53]. New assignments were created which use recursively defined lists and other structures. These structures are created using Java interfaces and classes which implement those interfaces. The authors claim that this reinforces the object-oriented programming style introduced earlier in the course and gives the students more experience

with recursion. This experience is concrete in the sense that recursive calls are invoked on existing objects (rather than relying on behind-the-scenes creation of activation records on the stack). Discussion of activation records can be avoided (or at least postponed). While the authors did not explicitly test for learning effect, they did report an improvement in student attitude toward the complexity of the course. Course evaluations, from one year to the next, contained lower overall course difficulty scores.

### 3.2.4 Algorithmic Approaches

Students are given a recipe or algorithm to follow for creating recursive functions. The most well known of these is the design recipe described in "How to Design Programs" [69]. The recipe consists of the following steps:

- perform Data Analysis and Design
- specify the Contract, Purpose, and Header
- create a set of Examples
- copy the appropriate Template
- define the Body
- run Tests

The appropriate template already contains prototype selector expressions, conditionals, and recursive calls. The student then must "fill in the blanks" when defining the body. This design recipe is a general approach to solving many programming problems. It is applicable

to producing recursive functions because it was created to be used by students programming in Scheme. Since Scheme is a functional programming language most programs require recursion.

A similar, but perhaps more specific, approach is described by Kay in [107]. These steps are shown below:

- Specify the function prototype
- Write comment (including pre-conditions, post-conditions, and return value
- Code the base case
- Think of a concrete example and closer second example
- Use the Force
- Solve concrete example and closer second example
- Code the recursive in an else statement

### 3.2.5 Visualizations

A number of visual representations have been developed for recursive procedures [71, 157, 162]. These ideas graphically represent the execution stack and activation records in some fashion [52].

The Tree Model (or Tree Diagrams) are described by Kruse in [112]. The main concept is to draw a n-ary tree which represents the execution of a recursive function. Each node in the tree represents one function invocation. Child nodes represent function invocations

performed by the parent invocation. The nodes can be annotated to show any parameter values. The diagram itself is static and only shows the one state of the execution.

Drawing a series of Tree Diagrams or providing a dynamic visualization of the diagram conveys more information. Further annotation of the tree nodes can be used to show function return values and other information. These extensions are called Activation Trees [95]. As in Tree Diagrams each node represents one specific function invocation. Each node contains the function name, a list of the parameter names and their current values, and a return value if appropriate. This approach extends Recursion Trees (described in [149] and elsewhere) by adding information from the runtime stack to the nodes.

Wu [162], describes a visualization program called SimRECUR. SimRECUR provides visualizations of three common conceptual models of recursion. These are the Stack Simulation Model, the Copies Model, and the Tree Model [112]. Each model is one way to describe how the computer behaves during execution of the recursive function. In addition to these, SimRECUR has a "codes" window which shows the code associated with the current invocation of the function as a series of stacked subwindows. A series of buttons allow the user to control the execution of the recursion. SimRECUR allows the student to write code in C, Pascal, or Basic. Recursive functions are "allowed to have one or two parameters." There is limited error checking. The visualizations were well received by students (as noted in an after use survey). The authors do not report the number of students who participated in the survey.

Dann et. al. [63] describes the use of the Alice programming environment [1] to visualize recursion. Alice allows the programmer to quickly create programs and stories which involve characters and other objects moving in a three-dimensional virtual world. Scripts

for the objects are written by dragging building block statements from menus into a programming window. While Alice does not truly provide recursion itself (at least at the time this paper was written), it can be used to write functions which can mimic recursive behavior. A timer is used to determine when the "mock" recursive function should be invoked again. The timer is also required to create a delay so that the animated characters have time to perform their defined motions (otherwise the animation may not be synchronous with the function execution). The authors report that students still encounter typical mistakes such as not creating base cases and not reducing the problem to a smaller subproblem. The graphical feedback provided by the visualization allows the student to find, understand, and fix any problems more quickly.

Franceschi introduces the idea of building movies to teach fundamental concepts including loops and recursion [73]. The visualization as presented is quite limited and consists of showing the currently executed line of code. There are a limited number of predefined movies (the movies are not generated or created by the user). The function parameter value is changed whenever the function is invoked during the recursion. There is no portrayal of the execution stack.

### 3.2.6 Dramatizations

In [52], Ben-Ari describes the concept of dramatizations for teaching recursion. The dramatizations are real-world problems which can be physically "executed" by students during a lecture. The dramatizations are carefully designed so that they correspond directly to a programming problem which can be solved using recursion. Three example dramatizations (and their corresponding programming problems) are presented:

- Open a present (Recognize balanced parenthesis)
- Build a chain (Compute factorial)
- Eat a chocolate bar (Search an array)

The dramatization algorithms are described in english-based pseudo code and are "executed" by students with physical materials. Recursive calls are represented as requests for assistance from neighboring students and by passing any and all necessary materials to those neighbors. The main advantages are: the dramatizations are fun, explanation of the program directly corresponds to the dramatization, and visualization of the stack is implicit and requires no additional explanation. The dramatizations are also physical concrete activities. It is believed that students are more likely to be engaged than when presented with more traditional problems (such as calculating factorial or checking for palindromes) [52].

### 3.2.7 Metaphors and Analogies

The Little Person metaphor proposes that computers are inhabited by "a large community of little people". Each of the little people is a "specialist at a particular procedure". Every little person has one set of instructions (also known as a script) but more than one person may have the same set of instructions. In addition, each person is wearing a vest with pockets. These pockets each have names and are used to hold the parameters of the person's script [94]. In the case of a recursive function, the little person hires another specialist (in the same procedure) when the recursion occurs. There is a slight problem with this portion of the metaphor. The metaphor assumes that all of the necessary little persons already exist. It also does not really help to describe how the execution is actually carried out

by the computer. Since the people always exist, they are never created or destroyed even though the people essentially represent activation records on the execution stack. A better metaphor would allow a little person to create clones of herself to perform the work of the recursive call. After the work is completed, the clone is then destroyed. This improved metaphor is closer to the creation of new activation records and the maintenance of the run-time stack that is performed by the computer as it executes the recursive function [66]. The Little People metaphor has been used by Logo teachers "for years" [94]. No studies have be carried out to determine the effectiveness of this approach.

#### 3.2.7.1 Revising the Little People Metaphor

We now describe a revision of the Little People metaphor that we believe is more effective and closer to the Copies Mental Model of recursion. Our revision still based on delegation but the actors are created and destroyed rather than pre-existing. We agree with the Little People metaphor in that delegation is an effective method to teach recursion which nearly anyone can understand. Most people are familiar with the concept of someone delegating a task to another person. Our method is based on this idea and presents recursion as a particular form of task delegation. We believe that this method improves understanding of how to program recursive procedures (especially by end-users and novice programmers). The work described in the remainder of this section was previously published in [66].

#### 3.2.7.2 Real World Delegation

Nearly everyone has had the experience of working for someone or being the "boss" themselves. In this context, work tasks are often delegated from the boss to the worker who will

complete the task. If necessary the boss gives the worker instructions describing how the task should be accomplished. The boss often waits for the workers to finish the task and expects some sort of feedback or result. To be complete, the boss recognizes a problem which he or she cannot solve. The problem is then decomposed into smaller problems. The boss identifies workers who are able to solve these smaller problems. The boss delegates these subproblems to the workers and waits for the results. When all of the workers have finished, the boss then can solve the original problem. This is the Divide-Conquer-Glue method as described in [158].

For example, consider a restaurant owner who has just closed the restaurant for the evening. There are several tasks which must be accomplished before going home: tables cleared, dishes must be washed, floors swept, and the kitchen cleaned. Suppose the owner has four workers. The owner assigns each worker a single task. Each worker does what is necessary to solve their individual problem (the one washing dishes may have to wash one or many dishes). There can be no further delegation because there is no one else at the restaurant. When the workers are finished, the overall task is finished and everyone can leave the restaurant and go home.

#### 3.2.7.3 Virtual World Delegation

The virtual world of computer programs is not the same as the real world. This profoundly affects the way that delegation can be used to solve problems. The boss wants to solve the original problem. The boss recognizes that he or she cannot solve this problem alone. Again the problem is decomposed into smaller problems. Things are different when assigning the tasks to workers however. The boss has additional powers which are not available in

the real world. Imagine the boss now possesses a magic wand. This wand has two powers: it can create new workers out of thin air and it can destroy them utterly and completely. Using the magic wand, the boss can create new workers and as many as desired. The subproblems can then be assigned as before. The boss not only decides how to assign the subproblems to the workers; the boss must also decide how many workers are needed or desired. When the subproblems are solved, the boss again has new options available. The created workers can be kept to be used again later or they can be easily destroyed using the wand.

Revisiting the restaurant example above, suppose instead that the restaurant described is a ”virtual” restaurant. There are still four tasks but the owner can now create as many workers as desired. In addition, each of the workers could create workers as well. For instance the owner may elect to create one worker for each of the four tasks. The worker washing dishes could then create hundreds of workers perhaps one for each dish which needs to be washed.

This is similar to the Little People metaphor [94]. This metaphor models computation as being carried out by a number of ”elves” each of which is an ”expert” in performing a particular task (such as printing a string). The metaphor is effective in describing how a computation is carried out but we believe it does not always aid novices in creating programs. The difficulty is that students may realize that there is an expert ”elf” for every task (including the program the student is trying to create). This misconception can lead the student to believe that no new code needs to be created. They just need to tell this ”elf” to perform its task. In other words, the student may believe that the computer already knows how to solve the student’s programming problem. The delegation framework avoids this

problem (and is more general) because it does not assume that the workers are ”intelligent”. It may be necessary that the boss inform each worker how the subtasks are to be solved.

#### 3.2.7.4 Virtual World Recursion

Recursion is a special case of virtual world delegation. Rather than create new workers, the boss creates clones of himself or herself as the workers. Since the clones are copies of the original boss they can create additional clones of themselves (clones of the clones). This can lead to problems if the clones do not stop this creation at some point. The cloning process stops when the problem that remains is simple enough that it can be easily solved (i.e. when the clone can solve the problem assigned to it without any requiring additional help from more clones of itself). Note that the clones have no additional knowledge or abilities beyond those that the boss possesses. The clones are exact replicas. Because of this, it is necessary that the boss know when this stoppage should occur and how to break each problem down into simpler problems.

In the restaurant example, this means that the owner must know how to perform each of the four tasks. However the owner does not have to do all of the work alone! Using the wand, the owner can create clones to accomplish portions of the work. In fact the owner can create as many clones as he or she desires. Each of the clones can be assigned any of the tasks. Each of the clones can also create more clones as desired to accomplish the assigned tasks (because the clones have the same power and abilities as the owner).

#### 3.2.7.5 Virtual World Iteration

As pointed out in [158] iteration is a special case of the Divide-Conquer-Glue method. It can also be specified in terms of delegation. For example, consider the problem of having to wash a pile of dirty dishes. The boss may decide to solve the problem alone (may not have a choice!). In this case, the boss still divides the problem into smaller subproblems; there is just no one else to which work can be assigned (in some sense iteration is the degenerate case of delegation). The problem is divided into one dirty dish and all of the remaining dirty dishes. The boss chooses a dish from the pile, washes it, and places the result in a pile of clean dishes. He then attacks the problem of washing the remaining dirty dishes in similar fashion. This problem can also be solved recursively if the boss has the ability to create clones. This different approach is described in the next section.

#### 3.2.7.6 Solving a Problem Recursively

There are several conditions which must be true in order to solve a problem using recursion. These conditions must be conveyed to the student in an understandable way. Usually the conditions are described as a set of base cases and a set of recursive cases: (Base Cases) There are simple problem instances which can be solved immediately. (Recursive Cases) The problem is reduced to smaller subproblems which are solved recursively.

When viewed in the light of delegation, these conditions are expanded to make the situation more clear to the student or end-user: (Base Cases) The boss must know how to solve simple problem instances without delegating to workers. Each of these simple problems is a base case of the recursion. (Problem Division) The boss must know how to divide a problem into smaller subproblems. This will move the process toward completion

and eventually force the recursion to end. (Create Clones) The boss must decide how many clones to create. Usually this is one clone for each subproblem. (Assign Subproblems) The boss must be able to assign the smaller subproblems to the clones. Usually this is one subproblem for each clone. (Combine Results) The boss must combine subproblem solutions into a solution for the overall problem.

Consider again the problem of washing a pile of dirty dishes. To solve this problem recursively, the boss can divide the problem into the two subproblems of washing one dish and washing the remaining dishes (this is just one example of how to divide this problem. There are many ways the problem could be divided). The boss could create two clones. The boss can then assign these two subproblems by assigning the single dish to one clone and the remaining dishes to the other clone. The first clone (which receives the single dish) must know how to wash it without creating more clones or the cloning will never stop (this also implies that the boss must know how to do this also because the clone and the boss have the same abilities). The second clone (which receives the remaining dishes) can perform the same process as the original boss. The second clone will create two clones and assign a single dish to the first clone and any remaining dishes to the second clone. Eventually these dishes are cleaned and combined into a single pile. Finally, the boss could combine the work of its two clones by placing the single clean dish (cleaned by the first clone) with the other remaining clean dishes (cleaned by the second clone or its clones) into one group of clean dishes.

We believe that this expanded list of conditions described above is easier for students to understand. It makes explicit the fact that there may be choices as to how many clones to create and how to assign the subproblems to them.

## 3.3 Summary

The theory of Mental Models is one of the current systems for describing not only how we mentally represent concepts, systems, and ideas but also how we reason about the world and events [105]. If this theory is correct, the creation of viable mental models is critically important.

Several studies have been conducted in order to understand and identify various Mental Models of boolean expressions and the concept of recursion [55, 85, 113, 147]. Mental Models of boolean expressions have been studied in order to investigate how we reason and think [105]. Some additional work has been done to improve representations of boolean expressions for programming and query languages [128]. For recursion particularly, considerable effort has been expended in creating new representations and dynamic visualizations in order to aid student understanding.

Our work builds on these ideas by using video games as a new form of visualization of boolean expressions and recursion. These visualizations are different than more traditional approaches in that the student no longer has complete control over visualization events. Student engagement is gained through game play and entertainment rather than asking the student questions about visualization events.

# Chapter 4

# A Game For Teaching Boolean Operators

## 4.1 Introduction

We present a simple video game designed to teach the player fundamental boolean operators. Two studies, one with high school students and one with college students, were conducted to test the effect of our game on learning outcomes. We report the results from both studies. Our results show a modest gain in learning outcomes. Unfortunately it appears that these gains are not the intended outcomes.

## 4.2 Not! The Simpsons Donut Drop

Our game, named "Not! The Simpsons Donut Drop", is a simple two dimensional side view video game. The player avatar is Homer Simpson. The player moves the avatar

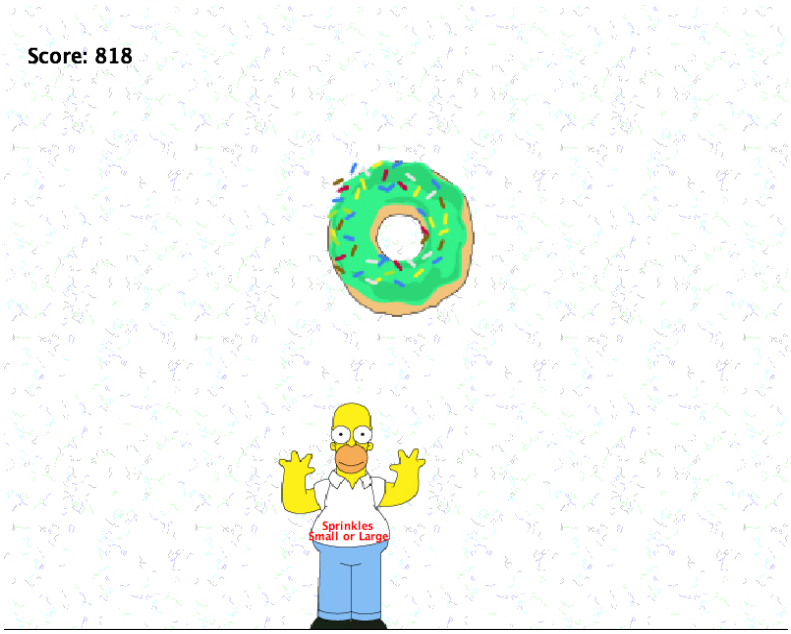


**Figure 4.1:** *Homer wants Donuts which have Sprinkles and are Small or Large in size*

from side to side at the lower edge of the game screen trying to catch falling donuts. The donuts have three properties: a flavor such as chocolate or lime, a size which can be small, medium, or large, and type of topping which can be sprinkles or none. At the beginning of each game level, Homer is redrawn so that his shirt shows which types of donuts he wants to catch. During each level (there are approximately twenty) a number of donuts fall from the top of the screen. To succeed the player must move Homer to catch the appropriate donuts. An example game level is shown in Figure 4.1. In this example, Homer should catch all donuts which have sprinkles and are either small or large size (the donut in the figure satisfies these criteria).

During the early levels, Homer's shirt shows the donut criteria in an english notation similar to match forms [130]. The criteria listed are conjuncts (i.e. all criteria are part of an overall AND expression). Each criteria is either a simple negation or the disjunction of a


**Figure 4.2:** *Homer wants Chocolate Donuts which are Not Small (English notation)*

series of criteria. The current version does not allow both negation and disjunction within the same conjunct.

In later levels, the criteria alternate between english and an equivalent Java syntax. This is to show the equivalent symbols in the two notations (i.e. "not" is equivalent to "!"). Note that the game does not use explicit notation for the AND operator because it is implicit in the match form notation. This notation was chosen for two reasons: it is compact which is important because there is limited space available on Homer's shirt and it has been shown to be effective with users [130]. After showing the player the various equivalencies, the remaining game levels use the Java notation exclusively.


**Figure 4.3:** *Homer wants Chocolate Donuts which are Not Small (Java language notation)*

## 4.3 Game Design and Implementation

We implemented the game in Java using the Greenfoot environment [16]. One of the main goals of this work is that the game should teach the meaning and concept of boolean operators not programming language syntax. Because of this goal, the game allows the use of different notations. This allows testing the effect of each notation on comprehension and learning. Comprehension results could help the designers of programming languages and end-user systems. Learning results could help instructors and educational game designers.

Another goal was to create games which can also serve as programming examples. All of the code is available to both instructors and students. This serves several purposes: instructors can modify the game to suit their own needs and the code can be used as an example of programming. One target audience is the instructors and students in an outreach

program, P4Games.org, for local Denver high schools. The P4Games curriculum uses games and art in order to increase interest in STEM disciplines and in attending college. The curriculum includes programming in Java using Greenfoot [28].

One method for teaching a particular notation (such as the syntax for Java) would be to start with english descriptions of the expressions and slowly introduce the symbols for the desired notation. This approach carries some risk because the student must play longer. If the game is not interesting this may not happen. It also requires a starting notation known to the student. A natural language such as english might not be ideal because it is often ambiguous but starting with something familiar to students outweighs this disadvantage.

## 4.4 Research Questions

We hypothesize that a simple video game can be an effective method for teaching boolean operations. In particular, we will test the following hypotheses:

- Playing Donut Drop will positively affect student knowledge of the AND operation.
- Playing Donut Drop will positively affect student knowledge of the NOT operation.
- Playing Donut Drop will positively affect student knowledge of the OR operation.
- Playing Donut Drop will improve knowledge of the Java representation of AND, NOT, and OR operations.

## 4.5 Methods

Two different groups of students participated in the assessment of the game. The first group consisted of thirty eight first year college students. The second group consisted of sixty high school students. Students volunteered to complete a pre-survey. The students then played the game for approximately 30 minutes. After playing, the students completed another survey identical to the pre-survey.

For the college students, the survey contained questions about the following areas: general questions about boolean variables, Java notational questions, and questions concerning the values of simple boolean expressions. The possible responses for the general questions and the Java notational questions were "true", "false", and "I Don't Know". The possible responses for the last group of questions were "One must be true", "One must be false", "Each must be true", "Each must be false", and "I Don't Know". All of the questions are shown in Section 4.6.

For the high school students, the survey consisted of a series of matching questions. The questions consisted of a simple boolean expression and a set of colored symbols. The students were asked to choose the symbols which satisfied the boolean expression. The matching questions were based on the study by Pane and our repetition of that study [130]. Some additional matching questions were added. The matching questions were followed by six additional questions to determine if the students could determine what conditions were necessary to make a simple boolean expression evaluate to true or false. The simple boolean expressions were:

- not X

- X or Y or Z
- X and Y and Z

In both studies, each response was classified as being "I Don't Know", incorrect, or correct. The tables of results show the number of responses in each of these categories for the pre-survey and the post-survey. The sums of the rows show the results for the pre-survey; the sums of the columns show the results for the post-survey. The tables also make it possible to determine how many participants in each category changed their responses from the pre-survey to the post-survey. For example in Table 4.5, four participants changed from incorrect on the pre-survey to the correct response on the post-survey. All of the questions are shown in Section 4.7.

The results from these two studies are described in the following sections. We denote the two different studies as the "College Study" and the "High School Study".

## 4.6 The College Study

Thirty eight college students volunteered to participate in the study. The participants were all students in the first quarter of a CS1 course. This study took place near the middle of the course so the students had already been exposed to boolean expressions and conditional statements. The participants completed a pre-survey and a post-survey which contained the same twenty-three questions. One person did not complete the post-survey and their responses were not included in the results.

### 4.6.1 Results

The first two survey questions concern the purpose of an If statement in a program and what values a boolean variable can hold. These first two questions were followed by a series of questions concerning Java syntax for the NOT, OR, and AND operations. The first eleven questions were:

1. If Statements Allow A Program To Make Decisions Based On Current Conditions
2. A Boolean Variable Can Only Store The Values Of True Or False
3. ! Represents NOT
4. ! Represents OR
5. ! Represents AND
6. || Represents NOT
7. || Represents OR
8. || Represents AND
9. && Represents NOT
10. && Represents OR
11. && Represents AND

For nearly every one of these first eleven questions most of the participants answered correctly on the pre-survey (only one or two participants answered incorrectly). For the one

or two participants that did not answer correctly on the pre-survey most of them answered correctly on the post-survey. At most, one participant answered incorrectly answers on the post-survey (although for the first question there was one incorrect answer and one answer of "I Don't Know").

The remaining twelve survey questions ask the participant to determine the conditions which make a given boolean expression true or false. These questions were:

1. If !X Is True Then X Must Be False
2. If !X Is True Then X Must Be True
3. If !X Is False Then X Must Be False
4. If !X Is False Then X Must Be True
5. If X || Y || Z Is True Then Each of X,Y, Z Must Be True
6. If X || Y || Z Is True Then Any of X,Y, Z Must Be True
7. If X || Y || Z Is False Then Each of X,Y, Z Must Be False
8. If X || Y || Z Is False Then Any of X,Y, Z Must Be False
9. If X && Y && Z Is True Then Each of X,Y, Z Must Be True
10. If X && Y && Z Is True Then Any of X,Y, Z Must Be True
11. If X && Y && Z Is False Then Each of X,Y, Z Must Be False
12. If X && Y && Z Is False Then Any of X,Y, Z Must Be False

The possible responses for these questions were “I Don’t Know”, “True”, or “False”. We present the results in Tables 4.1 through 4.12.

**Table 4.1:** *Question: “If !X Is True Then X Must Be False”*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don’t Know | Incorrect | Correct |
| I Don’t Know | 0 | 0 | 1 |
| Incorrect | 0 | 0 | 1 |
| Correct | 0 | 0 | 35 |

**Table 4.2:** *Question: “If !X Is True Then X Must Be True”*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don’t Know | Incorrect | Correct |
| I Don’t Know | 0 | 0 | 1 |
| Incorrect | 0 | 0 | 1 |
| Correct | 0 | 0 | 35 |

**Table 4.3:** *Question: “If !X Is False Then X Must Be False”*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don’t Know | Incorrect | Correct |
| I Don’t Know | 0 | 0 | 1 |
| Incorrect | 0 | 2 | 2 |
| Correct | 0 | 2 | 30 |

### 4.6.2 Analysis of College Study

The collected responses from all questions for the pre-survey and post-survey were analyzed using both Fisher’s Exact test and a paired t-test [62]. Results were considered significant if $p < 0.05$.

**Table 4.4:** *Question: "If !X Is False Then X Must Be True"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 0 | 0 | 1 |
| Incorrect | 0 | 1 | 3 |
| Correct | 0 | 2 | 30 |

**Table 4.5:** *Question: "If X || Y || Z Is True Then Each Of X,Y, Z Must Be True"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 0 | 0 | 0 |
| Incorrect | 0 | 2 | 4 |
| Correct | 0 | 1 | 30 |

#### 4.6.2.1 Basic Understanding and Java Notation

The first eleven survey questions concern basic understanding of conditional statements and the Java notation for boolean operators. Nearly all of the students answered these questions correctly on the pre-survey. In the few cases where one or two incorrect responses were present on the pre-survey, most of the incorrect responses were changed on the post-survey. There were no negative effects for any of questions. Since the students were in the second quarter of CS1 it is not surprising that they answered these questions correctly. The students learn about conditional statements and the Java notation for the basic boolean operators in the first quarter.

#### 4.6.2.2 NOT operator

The remaining questions concern knowledge of the operators themselves. The first set of four questions are concerned with the NOT operator. The students were asked questions

**Table 4.6:** *Question: "If X || Y || Z Is True Then Any Of X,Y, Z Must Be True"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 0 | 0 | 0 |
| Incorrect | 0 | 1 | 3 |
| Correct | 0 | 0 | 33 |

**Table 4.7:** *Question: "If X || Y || Z Is False Then Each Of X,Y, Z Must Be False"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 0 | 0 | 0 |
| Incorrect | 0 | 19 | 2 |
| Correct | 0 | 4 | 12 |

of the form: "If Not X is <expression-value> then X must be <result-value>" where <expression-value> and <result-value> are either "true" or "false". Each of the four questions is one of the possible combination of those values. For the first two questions the results (shown in tables 4.1 and 4.2) are identical: two students improved and answered correctly on the post-survey. All other students answered correctly on the pre-survey. This represents an improvement of 5.4%. This change is not statistically significant when analyzed using Fisher's Exact test (p = 1) or a paired t-test (p = 0.1061).

The results for the next two questions are shown in table 4.3 and table 4.4. For the first of these, three students improved. For the second question, four students improved. In both cases two students changed from correct on the pre-survey to incorrect on the post-survey. These changes are not statistically significant under either statistical test.

The game appears to have had a slight positive effect for the first two questions and little overall effect for the last two questions. We hypothesize that this is because the game

**Table 4.8:** *Question: "If X || Y || Z Is False Then Any Of X,Y, Z Must Be False"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 0 | 0 | 0 |
| Incorrect | 0 | 21 | 3 |
| Correct | 0 | 3 | 10 |

**Table 4.9:** *Question: "If X && Y && Z Is True Then Each Of X,Y, Z Must Be True"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 0 | 0 | 1 |
| Incorrect | 0 | 2 | 1 |
| Correct | 0 | 1 | 32 |

emphasizes the conditions for satisfying an expression rather than what conditions should be avoided. Since Homer's shirt specifies which donuts he desires, the player examines each donut to determine if it matches rather than which donuts fail to match. In other words, the player learns how to fulfill a criteria expression and not how to make the criteria expression fail. From this we conjecture that the game should be modified to account for this. One possibility would be to add objects or donuts which Homer must explicitly avoid in order to score points. Another possibility would be to specify criteria which only contain negations.

#### 4.6.2.3 OR Operator

The next four questions concern the "OR" boolean operator. Again the first two questions had nearly identical results. For the first question, four students improved and one student did worse. For the second question, three students improved and no students did worse.

**Table 4.10:** *Question: "If X && Y && Z Is True Then Any Of X,Y, Z Must Be True"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 0 | 0 | 1 |
| Incorrect | 0 | 3 | 3 |
| Correct | 0 | 1 | 29 |

**Table 4.11:** *Question: "If X && Y && Z Is False Then Each Of X,Y, Z Must Be False"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 0 | 1 | 0 |
| Incorrect | 0 | 23 | 2 |
| Correct | 0 | 2 | 9 |

These results are shown in tables 4.5 and 4.6. The three students who improved represent just over $8\%$ of the participant population. These improvements are not statistically significant under either test however.

The results from the next two questions show that the students have difficulty in determining what will make a disjunction false. Nineteen students answered the first question incorrectly on both the pre-survey and the post-survey. Two students improved and four students did worse. For the second question, twenty-one students answered incorrectly on both surveys. The results are shown in tables 4.7 and 4.8. The results for both questions are statistically significant when analyzed using Fisher's Exact test ($p = 0.00008$, and $p = 0.00016$ respectively) but not when using a paired t-test ($p = 0.427$ and $p = 1$ respectively).

**Table 4.12:** *Question: "If X && Y && Z Is False Then Any Of X,Y, Z Must Be False"*

| Pre-Survey | Post-Survey I Don't Know | Incorrect | Correct |
|---|---|---|---|
| I Don't Know | 0 | 1 | 0 |
| Incorrect | 0 | 19 | 2 |
| Correct | 0 | 5 | 10 |

#### 4.6.2.4 AND Operator

The next four questions concern the AND operator. As with the other operators, nearly all of the students answered the first two questions correctly. The results are shown in tables 4.9 and 4.10. A small number of students improved in each case. Two students improved on the first question and four improved on the second question. For each question one student did worse on the post-survey than on the pre-survey.

The students did less well on the next two questions. For the first of these (shown in table 4.11), twenty three students answered incorrectly on both surveys. Two students improved and two students did worse. For the second question (shown in table 4.12), nineteen students answered incorrectly on both surveys. Two students improved but five did worse.

### 4.6.3 Discussion

Many of the questions on the College Study survey were too easy for the college students. Most likely this is due to the fact that the game evaluation occurred late in the second quarter of a CS1 course and the students had already been taught boolean expressions and conditional statements. There were hardly any incorrect answers on the first two questions

or on any of the notational questions. These easier questions did not yield any significant results so it is impossible to tell if the game had any effect on knowledge of Java notation and our fourth hypothesis cannot be confirmed.

The results for the later questions (concerning values of variables in boolean expressions) were more useful. It is unclear why the negatively worded questions were answered incorrectly by so many participants. The wording of the questions may have confused many of the students. Although there was some minor improvements for two of these questions they were not statistically significant. Because of this, none of our hypotheses can be confirmed for the College Study.

It is interesting to note that more than half of the participants had difficulty determining what is required to make a disjunction or conjunction false (as seen in tables 4.7, 4.8, 4.11, and 4.12). Few participants had the same difficulty with the expressions involving simple negations (as seen in tables 4.3 and 4.4). These two observations fit the mental model theory of reasoning. In this theory, a mental model of a boolean expression only contains those necessary elements which ultimately make the expression evaluate to "true". It may be more difficult to reason about what makes an expression "false" because, in part, it seems to require more knowledge about the possible objects that exist in the "world" [105].

The game may have had little or no effect on these results because it does not explicitly include such examples. Homer's shirt specifies what donuts to catch, it never directly specifies what not to catch. The students may also have been confused by the negative wording of these questions. In either case, this warrants further study. New examples to explicitly cover these cases should be included in any future educational games. Special

care should be taken when teaching boolean operators to make sure that students understand what makes an expression false.

## 4.7 The High School Study

Sixty students who attend a local high school volunteered to participate in the second study. The students were all in courses supported by the P4Games project [28]. As in the College Study, the students were given a pre-survey, played the game for approximately 30 minutes, and then completed a post-survey. The game was modified to present questions using only "english" notation. No Java notation was included because the students had only begun to learn the Java language.

The questions on both surveys asked the participants to determine which of a set of colored symbols satisfy the specified boolean expression. Each symbol is either a triangle, circle, or square. Each symbol is also colored either red, blue, or green. The symbols used were:

- red triangle
- red circle
- red square
- blue triangle
- blue circle
- blue square

- green triangle
- green circle
- green square

### 4.7.1 Results

Table 4.13 shows the results of the first question, "Mark the objects that match blue", for all 60 participants. This question was included as a simple control to determine if the participants were truly paying attention while completing the surveys. Responses of participants who did not answer this question correctly (on either the pre-survey or the post-survey or both) were removed from the remaining analysis. Twenty three responses were removed for this reason, leaving a remainder of 37 responses.

**Table 4.13:** *Question: "Mark the objects that match blue"*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 10 | 2 |
| Correct | 11 | 37 |

The next question asked the participants to mark the symbols which match blue and circle. This question has more than one possible interpretation. If "and" is interpreted to be the AND operator, then only one symbol satisfies the expression: the blue circle. Under this interpretation, more than half of the participants answered incorrectly on the pre-survey. The results are even worse on the post-survey (only one person answered correctly). The results for this first interpretation are shown in table 4.14. These results are statistically significant using McNemar's chi-squared test with a p-value of 0.0025.

There is another interpretation. One could interpret "and" as a disjunction and match all the blue symbols along with all of the symbols which are circles. The results for this interpretation are shown in table 4.15. These results are statistically significant using McNemar's chi-squared test with a p-value 0.0036.

These two sets of responses indicate that our game is reinforcing the interpretation of "and" as a disjunction. This is interesting since our game does not explicitly use "and" to specify the types of donuts that Homer should catch.

**Table 4.14:** *Question: "Mark the objects that match blue and circle"*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 25 | 0 |
| Correct | 11 | 1 |

**Table 4.15:** *Question: "Mark the objects that match blue and circle" Alternate Interpretation*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 9 | 15 |
| Correct | 2 | 11 |

Since the next question contains "or" we expect the participants to do better than on questions which have "and" because our game explicitly represents OR and only implicitly represents AND. In the game the criteria are specified as a conjunction of a sequence of disjunctive phrases. The conjunction operators are not portrayed on Homer's shirt. There is little difference between the pre-survey and post-survey for this question however (this is shown in table 4.16). These results are not statistically significant.

**Table 4.16:** *Question: "Mark the objects that match square or green"*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 10 | 6 |
| Correct | 7 | 14 |

The next question asked the participants to choose the symbols that "match blue and green". If "and" is interpreted as the AND operator, none of the possible symbols should satisfy the expression. None of the participants used this interpretation. The results are shown in table 4.17.

**Table 4.17:** *Question: "Mark the objects that match blue and green"*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 37 | 0 |
| Correct | 0 | 0 |

Under the alternate interpretation of "and", the expression is satisfied by all of the blue symbols along with all of the green symbols. The results are quite different using this interpretation and are shown in 4.18. These results are not statistically significant, p=0.2888, but do show that the students consistently used this alternate interpretation.

**Table 4.18:** *Question: "Mark the objects that match blue and green" Alternate Interpretation*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 1 | 2 |
| Correct | 6 | 28 |

The next two questions test whether expressions that consist of two connected phrases will be interpreted differently from an identical single phrase expression (i.e. will "match red or circle" be interpreted differently from "match red or match circle"). We are interested if "or" and "and" will be interpreted as they were in the previous single phrase questions. We are particularly interested in whether participants will still interpret "and" as a disjunction. The first question tests these hypotheses for "or". The second question tests these hypotheses for "and". The results are shown in Tables 4.19 and 4.20. For the first of these questions, the results are essentially the same as the results for the earlier question, "Mark the objects that match square or green", shown in Table 4.16. For the second of these questions, the difference is interesting however. Students seem to be using the proper interpretation of "and" rather than interpreting it as a disjunction. Playing the game did not seem to affect these results however so the difference is in the wording of the questions. Adding the word "match" before each conjunct seems to be the key.

**Table 4.19:** *Question: "Mark the objects that match red or match circle"*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 13 | 7 |
| Correct | 5 | 12 |

**Table 4.20:** *Question: "Mark the objects that match blue and match circle"*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 3 | 7 |
| Correct | 8 | 19 |

The next two questions examine the participant's understanding of precedence between the NOT operator and the AND operator. Neither question uses parentheses so the participant decides what is the order of precedence. The participant may also use the alternate interpretation of "and". It is interesting to note that few participants answered the first question (shown in table 4.21) correctly assuming the standard interpretation of "and". The results are not much different when the alternate interpretation "and" is applied. This is shown in Table 4.22. Our game had little effect on either of these sets of results.

Many more answered the second of these questions correctly under the standard interpretation but not under the alternate interpretation as shown in Tables 4.23 and 4.24. Neither of these sets of results are statistically significant so again our game appears to have had little effect.

**Table 4.21:** *Question: "Mark the objects that match not red and square"*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 32 | 0 |
| Correct | 3 | 2 |

**Table 4.22:** *Question: "Mark the objects that match not red and square" Alternate Interpretation*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 35 | 2 |
| Correct | 0 | 0 |

We next asked the participants a question involving the NOT operator along with the OR operator. As in the previous two questions, no parentheses were used so the order of operations not clear. Table 4.25 shows that only one participant answered correctly on

**Table 4.23:** *Question: "Mark the objects that match square and not red"*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 5 | 7 |
| Correct | 11 | 14 |

**Table 4.24:** *Question: "Mark the objects that match square and not red" Alternate Interpretation*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 34 | 2 |
| Correct | 1 | 0 |

the post-survey assuming the interpretation that NOT will be given higher precedence than OR. Under the alternate interpretation, more students answered correctly as shown in Table 4.26. Once again neither set of results show a significant effect caused by our game.

**Table 4.25:** *Question: "Mark the objects that match not triangle or green"*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 36 | 1 |
| Correct | 0 | 0 |

The next question included parentheses in the boolean expression. The results are shown in Table 4.27. Again, nearly all of the participants answered incorrectly under the assumption that "and" would be interpreted as the AND operator. We show the results for the alternate interpretation where "and" is the OR operator in Table 4.28. More participants answered correctly but our game caused no improvement and the results are not statistically significant (p=0.1489 using McNemar's chi-squared test).

**Table 4.26:** *Question: "Mark the objects that match not triangle or green" Alternate Interpretation*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 6 | 3 |
| Correct | 8 | 20 |

**Table 4.27:** *Question: "Mark the objects that match not (triangle and red)"*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 36 | 0 |
| Correct | 0 | 1 |

The remaining six questions ask the participants to specify the conditions under which a simple expression involving variables evaluates to true or false. The game had little or no effect for the two questions involving the NOT operator (shown in tables 4.29 and 4.30). For both questions approximately half the participants answered correctly.

There was some improvement for the next question. Five additional participants answered correctly on the post-survey (shown in table 4.32). This improvement is not statistically significant (p=0.1306). It is interesting that this question is one of the four that was difficult for many of the participants in the previously described College Study.

The results are similar for the question shown in Table 4.33. Six people improved and answered correctly on the post-survey. These results are not statistically significant (p =0.0771).

There was no significant change for the last question. These results are shown in Table 4.34. As can be seen in the table, five participants changed from a correct response

**Table 4.28:** *Question: "Mark the objects that match not (triangle and red)" Alternate Interpretation*

| | Post-Survey | |
|---|---|---|
| Pre-Survey | Incorrect | Correct |
| Incorrect | 6 | 3 |
| Correct | 9 | 19 |

**Table 4.29:** *Question: "If not X is true then X must be..."*

| | | Post-Survey | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 5 | 1 | 0 |
| Incorrect | 1 | 4 | 3 |
| Correct | 0 | 5 | 18 |

on the pre-survey to incorrect on the post-survey. These changes are not significant with p=0.4497.

### 4.7.2 Analysis of High School Study

The collected responses from all questions for the pre-survey and post-survey were analyzed using McNemar's chi-squared test [62]. Results were considered significant if $p < 0.05$. Only one of the questions showed a significant effect caused by our game. This question, "Match the objects that match blue and circle" showed a statistically significant negative effect when the results are interpreted using the AND operator. When the question is interpreted using the OR operator for "and" then the game appears to have had a positive effect. Some of the other results show minor improvements in the student responses but none of these were statistically significant.

**Table 4.30:** *Question: "If not X is false then X must be..."*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 4 | 2 | 1 |
| Incorrect | 1 | 7 | 3 |
| Correct | 1 | 2 | 16 |

**Table 4.31:** *Question: "If X or Y or Z is true then..."*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 3 | 1 | 0 |
| Incorrect | 3 | 16 | 5 |
| Correct | 0 | 5 | 4 |

### 4.7.3 Discussion

It appears that our game had little or no effect with the High School participants. In the case where it did have an effect, it seems to reinforce the interpretation of "and" as the OR operator. This is the opposite effect of what was intended. The game was intended to improve the understanding of the mathematically defined boolean operators.

It is interesting to note that for the last two questions, there was some slight improvement. One of these, "If X or Y or Z is false then...", is a question that was incorrectly answered by more than half of the participants in the College Study. This effect could be random however.

There are a number of possibilities why our game had no effect in this study. First, the students may not have had the inclination to pay attention to the survey questions or that the questions were not understood. Some of the results of the first survey question, "Mark the objects that match blue", bear this out: one sixth of the students answered incorrectly

**Table 4.32:** *Question: "If X or Y or Z is false then..."*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 3 | 1 | 1 |
| Incorrect | 1 | 16 | 5 |
| Correct | 0 | 1 | 9 |

**Table 4.33:** *Question: "If X and Y and Z is true then..."*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | I Don't Know | Incorrect | Correct |
| I Don't Know | 3 | 0 | 1 |
| Incorrect | 1 | 9 | 6 |
| Correct | 0 | 1 | 16 |

on both surveys and another sixth switched to an incorrect answer on the post-survey. In more than one case, a participant marked all of the objects which were not blue.

Second, our game does not allow the student to reflect on what is happening and the playing time may have been too short. The students only played for one half hour. A study which allows more playing time is warranted in order to test this possibility.

Third, it may be that the feedback provided by the game is not strong enough to be effective. The game only notifies the player whether the donut satisfies the criteria or not. The game may need to specifically portray why a donut does or does not meet the desired criteria.

It may also be that information learned during game play does not transfer well to the survey questions. Perhaps the surveys should be rewritten so that the questions are about choosing donuts that satisfy a specified expression. This would serve to "narrow the gap" between the game and the survey questions.

**Table 4.34:** *Question: "If X and Y and Z is false then..."*

| Pre-Survey | Post-Survey I Don't Know | Incorrect | Correct |
|---|---|---|---|
| I Don't Know | 3 | 2 | 0 |
| Incorrect | 4 | 20 | 2 |
| Correct | 1 | 4 | 1 |

Another issue is that our game does allow the possibility of "false positives". It may not be possible for the avatar to reach every appropriate donut before it reaches the bottom of the screen. In some cases the avatar may hit a donut which does not match the avatar's current donut criteria. In other words, the player may hit a donut unintentionally. This presents a problem with the analysis: How can we tell if the player made a mistake because of the game or because of a lack of understanding? Did the avatar hit a donut because the player knows it is correct to do so or was it struck accidentally?

Finally it may be that the game is simply not engaging enough. The game uses a well known cartoon character but the overall design of the game is somewhat monotonous. There is no overarching goal and the levels are all essentially the same. We did notice that competition seemed to play a role in engaging a number of the students however. Several students insisted on playing the game again in order to improve their score and beat their peers. Instructors also observed students playing the game several weeks after our surveys were conducted.

## 4.8 Summary

In the College Study, the game had little or no effect for most of the questions (in any case most of the results were not statistically significant). The college students had the most difficulty with the negatively worded questions. More than half of the participants could not correctly determine the conditions under which a conjunction evaluates to false. Similar results were observed for simple disjunctions.

The lack of effect in the College Study may have occurred because the students already knew the answers to the questions on the survey. The students were in the second quarter of a introductory computer science sequence (CS1). In the previous quarter, the students were taught the concept of conditional statements and shown the simple boolean operators. Because of this, the survey questions may have been too simple for these students.

It is interesting to note the difficulty the college students had with the negatively worded questions. It would be interesting to investigate whether this is caused by the questions themselves (i.e. if they were reworded would the students have done better?) or is it caused by a misconception as to what makes a disjunctive or conjunctive expression false?

In the High School Study, the game also had little effect. The one exception is the effect on the results of the second question. For this question, the game appears to reinforce the interpretation of "and" as the disjunction operation. This was not the intended effect of the game. There were some minor improvements for other questions but none were statistically significant.

# Chapter 5

# A Game For Teaching Concepts Of Recursion

## 5.1 Introduction

We present a simple video game which acts as an effective visualization of recursion. We then show the results of using this game-based approach to augment the traditional lecture learning of recursion. We report the effect on student attitude and knowledge of recursion when the game is used in a CS1 course. Our results show a significant improvement in attitudes towards recursion and a modest improvement in recursion principle knowledge.

## 5.2 Recursive Breakout

Our game is a modification of the well-known video game Breakout (also known as Bricks). In Breakout, the player controls a simple paddle near the bottom of the play field to hit a

ball toward a group of blocks near the top. If the ball hits a block, then block is removed from the play field. When all blocks are removed a new level is started, with a new set of blocks, and game play continues. After striking a block or the top, left, or right boundary, the ball bounces and changes direction in an appropriate manner. If the ball moves past the paddle at the bottom then the ball is lost and the number of remaining balls is deducted by one. The player can prevent this by hitting the ball with the paddle. When the number of remaining balls reaches zero the game ends.

There are many variations of the basic Breakout game. The variations usually involve different types of blocks. For instance, there may be blocks which must be hit by the ball multiple times before they are destroyed or cause the ball to change velocity. Other variations allow the possibility of multiple balls on the screen or for the paddle to temporarily catch the ball and release it from another position on the play field.

In our game, there are two types of blocks: normal and recursive. Normal blocks behave as in the original game (when a normal block is hit by the ball, points are scored, the block disappears, and the ball motion is altered). Whenever a recursive block is hit by the ball, the current game level is immediately suspended and a new (recursive) level is created. This level creation is shown visually as an animation where the new game level grows outward from the recursive block until it reaches its full size. The new level has its own set of blocks, a paddle, and a ball. The new recursive level is played until either all game lives are lost (ending the game completely) or the level is completed. If the recursive level is completed then the previous game level is resumed at the point at which it was suspended. The new recursive game level is created so that it contains fewer recursive blocks than the level it was created from thus ensuring that the recursion would terminate. Note that this is

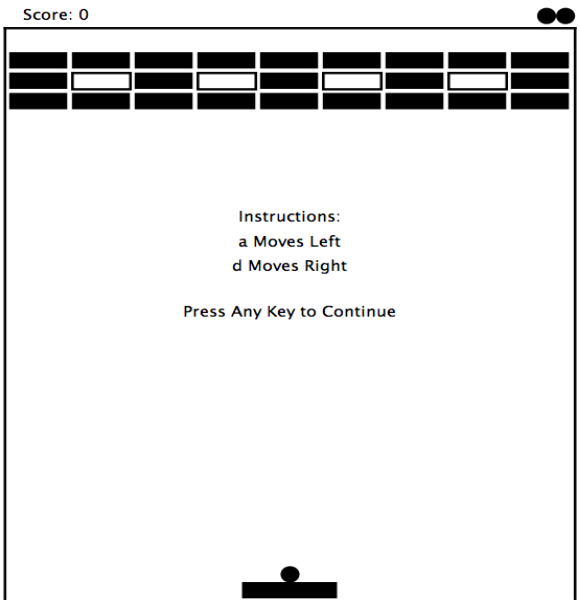


**Figure 5.1:** *Recursion Game Initial Screen*

not a requirement: It only matters that eventually a level is created which has no recursive blocks. To aid the instructional purpose we have included the ability to pause the game, restart the game, and adjust the ball speed. The initial game screen is shown in Figure 5.1.

All game levels are visible to the player as a superimposed stack of levels. Two examples are shown in Figures 5.2 and 5.3. In Figure 5.2 two levels of recursion are shown. Figure 5.3 depicts three levels of recursion. In these figures, the recursive bricks are colored white and the normal bricks are colored black. The upper most paddle is currently active. The others are the paddles for the previously suspended and unfinished levels.

In summary, the game is a visual metaphor for the execution of a recursive procedure. The game levels represent the current and past execution environments of an executing recursive procedure. Recursive blocks represent the potential of a recursive procedure call.

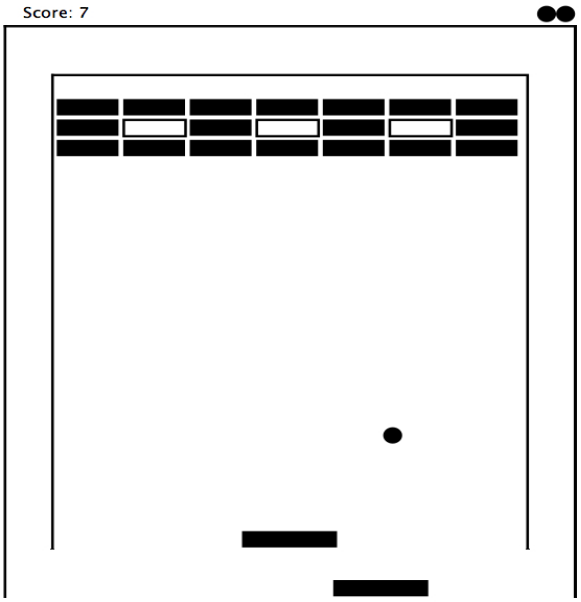


**Figure 5.2:** *Recursion Game showing two levels of recursion*

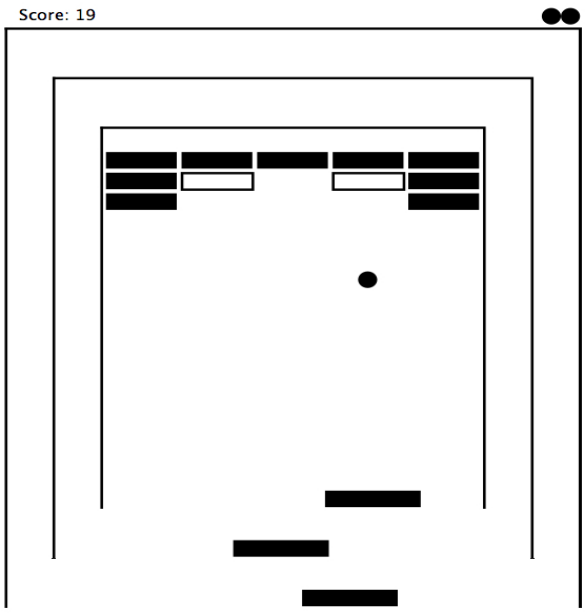


**Figure 5.3:** *Recursion Game showing three levels of recursion*

## 5.3 Game Design and Implementation

Our main design goal was to create a video game that serves as a visualization of recursion while at the same time being fun and engaging. The game should serve both students and instructors (i.e. instructors should be able to use the game as part of a course lecture on recursion). Further, the game should portray the current function invocation and all parent function invocations. This portrayal should include representation of the current executing environment, and both suspension and resumption of execution of an executing environment. A secondary design goal was simplicity of play so that the student does not need to concentrate much on game-play, but rather is able to focus on what is happening in the game as pertains to recursion. Another secondary goal was that the code of the game itself would be simple and understandable by students.

## 5.4 Research Questions

Our main hypothesis is that a simple video game based on the game of Breakout will increase understanding of and create more positive attitudes toward the concept of Recursion. These include whether or not recursion is believed to be complex or is difficult to understand. We also consider student confidence of their overall understanding, of how recursive functions are called, and the concept of base cases. Specifically we will test the following hypotheses:

- Playing Recursive Breakout will positively affect student beliefs about their level of understanding of recursion.

- Playing Recursive Breakout will positively affect student beliefs about the complexity and difficulty of recursion.
- Playing Recursive Breakout will improve general student knowledge of recursive functions.
- Playing Recursive Breakout will improve student knowledge of the purpose and structure of recursive functions (divide, conquer, and glue).
- Playing Recursive Breakout will improve student knowledge of the relationship between recursive functions and other functions (whether they must return a value and whether they can call other functions).
- Playing Recursive Breakout will improve student knowledge of base cases (including how many are required).
- Playing Recursive Breakout will improve student knowledge of termination requirements for recursive functions.

We wish to determine if the interaction with the game will provide the engagement necessary to make the visualization effective as a learning tool. This allows us to determine if the game does in fact meet the conditions described by Hundhausen [100] and Naps [127] as discussed in Section 3.2.

Our work aims to address the following two main questions:

- Will playing our proposed recursion video game, combined with a lecture about the game, change the participant's attitudes toward recursion in a positive manner; and

- Will it also improve the level of the participant's recursion knowledge.

In addition, we also wish to determine the participant's attitude toward the game itself.

## 5.5 Methods

We assessed learning efficacy by recruiting students to play the game and complete pre- and post-surveys. Our surveys measure changes in both attitude and knowledge. Twenty three students volunteered to participate in the survey. Two participants did not complete the post survey. One participant completed the post survey but did not complete the pre-survey. The data from these three participants was not included in the analysis.

The students were in the third quarter of a college introductory computer science course sequence (CS1). All had been exposed to recursion in the previous quarter, but many appeared to continue to have a weak understanding of recursion.

The surveys consisted of three parts: demographic information, student attitude, and knowledge questions. The demographic information consists of name (for identification purposes only), age, type of degree (Bachelor of Science or Arts), and intended Major of study.

Attitude information was collected to determine if playing the game changed beliefs and attitudes about recursion. Responses for these questions consisted of a five value Likert scale and the responses ranged from strongly disagree to strongly agree. Some of the questions investigate beliefs about confidence and understanding while others consider beliefs about recursion itself (e.g. Does the participant believe that recursion is complex or difficult to understand?). The first group of questions concerns the participant directly and

specifically. The second group is somewhat more independent of the participant. Three additional attitude questions were included in the post-survey to determine attitude toward the game itself. These questions asked if the students thought playing the game was helpful for understanding recursion, if they thought the game was fun to play, and if the game should be used in future courses.

The knowledge section consisted of a series of definitional and descriptive questions concerning basic concepts of recursion. The questions can be divided into several groups: those concerning base cases, those concerning the overall execution of a recursive function, and those regarding other properties of a recursive function. For the first group, the questions concerned the role, number, and necessity of base cases. The second group included questions regarding the aspects of the divide, conquer, and glue structure of recursive functions. The last group includes questions about calling other functions and returning values. The possible responses were "True", "False", or "I don't know".

The students were given the initial survey and then played the Recursive Breakout game for approximately 30 minutes. Following game-play students participated in a discussion of the purpose of the game and how it was connected to recursion. Afterward the students completed the post-survey.

The discussion was included in order to connect elements of the game with the various concepts of recursion. The discussion included the names of the various concepts (e.g. base cases) and how the game represents those concepts (e.g. game levels without recursive bricks). Presenting these connections was necessary because the game does not include or present the names of the different concepts of recursion. The students were also asked to consider whether the game would eventually end and how that could be determined.

The discussion also included a recursive description of the structure of a binary tree and a recursive algorithm to traverse a binary tree.

## 5.6 Results

### 5.6.1 Data for Attitude Questions

We first examine the data resulting from the six attitude questions. The data from each question are presented in Tables 5.1 through 5.6. The rows represent responses on the pre-survey. The columns represent the responses on the post-survey. From the table it is possible to determine how many participants changed their responses for each of the possible responses. For example, in Table 5.1, the second row shows that 2 participants responses' were "Disagree" on the pre-survey and of those one participant responded "Strongly Disagree" and one responded "Neutral" on the post-survey.

If the question is worded in a "positive" manner then the responses which appear above the the main diagonal of the table are from those participants who agree more strongly on the post-survey than on the pre-survey. These entries indicate a positive effect on that participant. Responses below the main diagonal of the table represent a negative effect. For "negatively" worded questions, the entries above the main diagonal represent a negative effect and the entries below the main diagonal represent a positive effect. This is the opposite of "positively" worded questions. For either type of wording, entries which are on the main diagonal of the table represent participants that did not change their responses from the pre-survey to the post-survey.

The first two questions were designed to determine whether the participants believe recursion is complex and difficult to understand. The results are shown in table 5.1 and table 5.2. For the question "Recursion is Complex", 3 participants increased their responses by one level toward the "Strongly Agree" end of the scale. Six participants did not change their responses (4 remained at "Neutral" and 2 at "Agree"). The remaining 11 participants decreased their responses by at least one level toward the "Strongly Disagree" end of the scale. It is interesting that only 2 participants "Strongly Disagree" that recursion is complex on the post-survey. This is only 10% of the participant population.

**Table 5.1:** *Results of question: "Recursion Is Complex"*

| | Post-Survey | | | | |
|---|---|---|---|---|---|
| Pre-Survey | Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |
| Strongly Disagree | 0 | 0 | 0 | 0 | 0 |
| Disagree | 1 | 0 | 1 | 0 | 0 |
| Neutral | 0 | 1 | 4 | 1 | 0 |
| Agree | 1 | 2 | 0 | 2 | 1 |
| Strongly Agree | 0 | 0 | 2 | 4 | 0 |

For the question "Recursion is Difficult To Understand", 2 participants increased their responses toward the "Strongly Agree" end of the scale. One participant increased by 2 levels (from "Disagree" to "Agree") and the other by one level (from "Neutral" to "Agree"). Seven participants did not change their responses. The other 12 participants decreased their responses by at least one level toward the "Strongly Disagree" end of the scale. Of these 12, two participants changed by more than one level. As in the previous question, only 2 participants "Strongly Disagree" that recursion is difficult to understand on the post-survey.

**Table 5.2:** *Results of question: "Recursion Is Difficult To Understand"*

| | Post-Survey | | | | |
|---|---|---|---|---|---|
| Pre-Survey | Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |
| Strongly Disagree | 0 | 0 | 0 | 0 | 0 |
| Disagree | 1 | 2 | 0 | 1 | 0 |
| Neutral | 0 | 4 | 1 | 1 | 0 |
| Agree | 1 | 1 | 2 | 4 | 0 |
| Strongly Agree | 0 | 0 | 2 | 0 | 0 |

One possible conclusion to draw from these two tables is that the participants believe that recursion is complex and difficult to understand, but after playing the game and discussing it they felt that recursion is less complex and difficult to understand than before the experience.

We now examine the results from the question "I Understand the Concept of Recursion" shown in table 5.3. One participant changed from "Strongly Agree" to "Agree". Eight participants did not change their responses. The remaining 11 participants increased their responses toward the "Strongly Agree" end of the scale. It is interesting to note that all participants responded "Agree" or "Strongly Agree" on the post-survey (exactly split between the two levels).

One possible conclusion is that before the game most participants believed they understand recursion, but after the game all believed they understand recursion and in general felt more confident in their understanding.

The results of the question "I Know What a Recursive Function Is" are shown in table 5.4. The results are very similar to the results of the previous question. Half of the participants did not change their responses (of these ten, 5 responded "Agree" and 5 responded

**Table 5.3:** *Results of question: "I Understand the Concept of Recursion"*

| | Post-Survey | | | | |
|---|---|---|---|---|---|
| Pre-Survey | Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |
| Strongly Disagree | 0 | 0 | 0 | 0 | 0 |
| Disagree | 0 | 0 | 0 | 3 | 0 |
| Neutral | 0 | 0 | 0 | 1 | 1 |
| Agree | 0 | 0 | 0 | 5 | 6 |
| Strongly Agree | 0 | 0 | 0 | 1 | 3 |

"Strongly Agree"). The other half increased their responses by at least one level. Note that the 3 participants who responded "Disagree" on the pre-survey all moved to "Agree" or "Strongly Agree".These responses further support the conclusion that after playing the game participants believe that they had a better understanding of recursion.

**Table 5.4:** *Results of question: "I Know What A Recursive Function Is"*

| | Post-Survey | | | | |
|---|---|---|---|---|---|
| Pre-Survey | Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |
| Strongly Disagree | 0 | 0 | 0 | 0 | 0 |
| Disagree | 0 | 0 | 0 | 2 | 1 |
| Neutral | 0 | 0 | 0 | 3 | 0 |
| Agree | 0 | 0 | 0 | 5 | 4 |
| Strongly Agree | 0 | 0 | 0 | 0 | 5 |

The results of the question "I Understand How Recursive Functions are Called" are shown in table 5.5. These results are very similar to the results from the previous two questions. One participant lowered their response from "Strongly Agree" to "Agree". Eight participants did not change their responses. The remaining 11 participants increased their responses by at least one level. Again note the increases for the two participants who

responded “Disagree” on the pre-survey. These responses further support the previous conclusion. Based on the results, we conjecture that the game and lecture may be most effective for the participants who have the least faith in their knowledge (i.e. those that responded “Disagree” or “Strongly Disagree” on the pre-survey).

**Table 5.5:** *Results of question: “I Understand How Recursive Functions Are Called”*

| | Post-Survey | | | | |
|---|---|---|---|---|---|
| Pre-Survey | Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |
| Strongly Disagree | 0 | 0 | 0 | 0 | 0 |
| Disagree | 0 | 0 | 0 | 1 | 1 |
| Neutral | 0 | 0 | 0 | 3 | 1 |
| Agree | 0 | 0 | 0 | 6 | 5 |
| Strongly Agree | 0 | 0 | 0 | 1 | 2 |

The results of the last attitude question are shown in table 5.6. Twelve of the participants did not change their response to the question “I Understand the Concept of Base Cases”. Three participants lowered their responses (one from “Agree” to “Disagree” and two from “Strongly Agree” to “Agree”). The remaining five participants increased their responses. Again the most striking improvements occur for the participants who responded “Strongly Disagree” and “Disagree” on the pre-survey. Three out of four participants answered “Agree” or “Strongly Agree” on the post-survey for this question. One possible conclusion to draw is that the game does not sufficiently represent recursion base cases in an explicit manner.

**Table 5.6:** *Results of question: "I Understand The Concept Of Base Cases"*

| | Post-Survey | | | | |
|---|---|---|---|---|---|
| Pre-Survey | Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |
| Strongly Disagree | 0 | 0 | 0 | 1 | 0 |
| Disagree | 0 | 1 | 1 | 1 | 1 |
| Neutral | 0 | 0 | 2 | 1 | 0 |
| Agree | 0 | 1 | 0 | 4 | 0 |
| Strongly Agree | 0 | 0 | 0 | 2 | 5 |

### 5.6.2 Data for Post-Survey Attitude Questions

Three additional attitude questions were asked as part of the post-survey. The results from these questions are shown in tables 5.7, 5.8, and 5.9. These questions asked whether the participants believed that the game was helpful for understanding recursion, whether the game was fun, and whether the game should be used in future courses.

Over half of the participants agreed that the game helped them understand recursion. Eighteen participants thought that the game was fun to play and seventeen believed that the game should be used in courses. It is possible that the participants believe the game should be used in courses because rather do anything or play any game rather than to listen to another boring lecture. When these results are considered in the context of the game as a visualization of recursion, we surmize that they show that the students were engaged in the visualization through game play. Analysis of the knowledge questions is required in order to test this question.

**Table 5.7:** *Playing The Recursive Breakout Game Helped Me Understand Recursion*

| | Post-Survey | | | | |
|---|---|---|---|---|---|
| | Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |
| Responses | 2 | 2 | 3 | 10 | 3 |

**Table 5.8:** *The Recursive Breakout Game Is Fun*

| | Post-Survey | | | | |
|---|---|---|---|---|---|
| Pre-Survey | Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |
| Responses | 0 | 1 | 1 | 8 | 10 |

### 5.6.3 Data for Knowledge Questions

We present survey results for thirteen questions concerning recursion knowledge. Possible responses were "True", "False", and "I Don't Know". In each of the following tables, the correct response is listed first, the incorrect response is listed second, and "I Don't Know" is shown last. For several questions, some of the participants did not respond. Those responses are included in the results when they occurred as an "I Don't Know" response.

We present both results for each question in the following tables. The first row contains entries for those who answered correctly on the pre-survey. The first entry in this row represents those whose answer did not change. The remaining entries represent those who changed to an incorrect or "I Don't Know" answer on the post-survey which shows a negative effect. The second row contains entries for those who answered incorrectly on the pre-survey. The first column entry in this row represents participants who changed the answer from incorrect in the pre-survey to correct in the post-survey which is a positive effect. The third entry represents a (possible) positive improvement as the participant is now

**Table 5.9:** *The Recursive Breakout Game Should Be Used In Courses To Teach Recursion*

| | Post-Survey | | | | |
|---|---|---|---|---|---|
| Pre-Survey | Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree |
| Responses | 0 | 0 | 3 | 9 | 8 |

unsure of what the correct answer is and may now recognize that their pre-survey response was incorrect. The first column entry in the third row represents participants who answered "I Don't Know" and answered correctly on the post-survey. As before any entries on the main diagonal of the tables represents participants that did not change their response from the pre-survey to the post-survey.

For the question, "A Recursive Function is a Function Which Calls Itself", nearly all participants (19 out of 20) responded with the correct answer ("True") on the pre-survey. The remaining participant who responded "I Don't Know" changed their response to "True" on the post-survey. This is shown in table 5.10.

**Table 5.10:** *Question: "A Recursive Function Is A Function Which Calls Itself"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | True | False | I Don't Know |
| True | 19 | 0 | 0 |
| False | 0 | 0 | 0 |
| I Don't Know | 1 | 0 | 0 |

The next two questions are designed to determine if the participants understand the divide, conquer, and glue characterization of recursive functions described by Turbak et. al. in [158]. This characterization was discussed in the lecture which followed playing the game. Fifteen out of twenty participants answered the first of these questions, "A Recursive

Function Divides A Problem Into Simpler Problems", correctly on the pre-survey and did not change their response on the post-survey. Two changed from "True" to "False". One participant changed from "False" to "I Don't Know". Two changed from "I Don't Know" to "True". In summary, two participants changed from correct to incorrect. Two others changed from unsure to correct. One changed from incorrect to unsure.

**Table 5.11:** *Results of question: "A Recursive Function Divides A Problem Into Simpler Problems"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | True | False | I Don't Know |
| True | 15 | 2 | 0 |
| False | 0 | 0 | 1 |
| I Don't Know | 2 | 0 | 0 |

For the second of the two questions, "A Recursive Functions Combines the Solutions of Simpler Problems into the Solution of the Original Problem", thirteen participants responded with the correct answer ("True") on the pre-survey. One participant answered incorrectly on the pre-survey and did not change their response on the post-survey. Five out of six who responded "I Don't Know" on the pre-survey changed to the correct answer on the post-survey while the sixth again responded "I Don't Know".

**Table 5.12:** *Results of question: "A Recursive Function Combines The Solutions Of Simpler Problems Into The Solution Of The Original Problem"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | True | False | I Don't Know |
| True | 13 | 0 | 0 |
| False | 0 | 1 | 0 |
| I Don't Know | 5 | 0 | 1 |

It would appear that the game had little effect on the knowledge of the concept of problem division (knowledge of the "Divide" concept) but did affect the knowledge concerning the concept of combining solutions of subproblems into the larger solution (the "Glue" concept).

The next question, "A Recursive Function Cannot Call Other Functions", is designed to determine if the participants understand that recursive functions are not "special" and they can invoke other functions including other recursive functions. The results are shown in table 5.13. For this question it appears that the game may have had a negative effect. Four participants who answered correctly on the pre-survey changed their answers. Three changed their answer to "True" and one changed to "I Don't Know". On the other hand, three participants changed from "I Don't Know" to the correct answer on the post-survey.

**Table 5.13:** *Question: "A Recursive Function Cannot Call Other Functions"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | False | True | I Don't Know |
| False | 13 | 3 | 1 |
| True | 0 | 0 | 0 |
| I Don't Know | 3 | 0 | 0 |

The next question investigates whether participants believe that recursive functions must return a value. While only 12 participants answered correctly on the post-survey this represents an improvement over the 7 participants who answered correctly on the pre-survey.

The next seven questions concern the concept of base cases and their role in recursion. Five participants answered the first of these, "A Recursive Function Must Have Exactly One Base Case", correctly on the pre-survey. Only 8 participants responded correctly on

**Table 5.14:** *Results of question: "A Recursive Function Must Return A Value"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | False | True | I Don't Know |
| False | 6 | 1 | 0 |
| True | 4 | 6 | 0 |
| I Don't Know | 2 | 1 | 0 |

the post-survey. It is also interesting that on the post-survey, 8 participants responded "I Don't Know" and 5 answered incorrectly. This appears to indicate once again that the game does not portray base cases in an effective manner.

**Table 5.15:** *Results of question: "A Recursive Function Must Have Exactly One Base Case"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | False | True | I Don't Know |
| False | 5 | 0 | 1 |
| True | 2 | 2 | 2 |
| I Don't Know | 1 | 3 | 4 |

Eighteen participants answered the next question, "A Recursive Function Must Have At Least One Base Case", correctly on the post-survey. Sixteen correctly answered "True" on the pre-survey, and of these one changed their answer to "I Don't Know" on the post-survey. One participant changed their incorrect answer of False to True. Two participants changed from "I Don't Know" to the correct answer, and one participant responded "I Don't Know" in both the pre-survey and post-survey. Far fewer people answered the previous question correctly.

Eight participants answered the next question, "A Recursive Function Must Have At Least One Base Case", correctly on the post-survey. Seven correctly answered "False" on the pre-survey, and of these one changed their answer to "I Don't Know" on the post-

**Table 5.16:** *Results of question: "A Recursive Function Must Have At Least One Base Case"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | True | False | I Don't Know |
| True | 15 | 0 | 1 |
| False | 1 | 0 | 0 |
| I Don't Know | 2 | 0 | 1 |

survey. One participant changed their incorrect answer of "True" to the correct answer of "False". One participant changed from "I Don't Know" to the correct answer, and four participants changed from "I Don't Know" to the incorrect answer on the post-survey.

**Table 5.17:** *Results of question: "A Recursive Function Cannot Have More Than One Base Case"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | False | True | I Don't Know |
| False | 6 | 0 | 1 |
| True | 1 | 2 | 0 |
| I Don't Know | 1 | 4 | 5 |

Results for the third question, "A Base Case is a Subproblem with a Recursive Solution", are shown in table 5.19. Fourteen responded correctly on the post-survey. Only 4 participants responded incorrectly on the post-survey. Seven answered incorrectly and 7 responded "I Don't Know" on the pre-survey. Three participants who answered correctly on the pre-survey changed their answer on the post-survey however.

The last question concerning base cases, "A Recursive Function With Base Cases Will Always Finish", was answered correctly by seven participants on the post-survey. Seven answered incorrectly on the post-survey. Nine participants answered correctly on the pre-survey.

**Table 5.18:** *Results of question: "A Base Case Is A Subproblem With A Non-Recursive Solution"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | True | False | I Don't Know |
| True | 9 | 0 | 1 |
| False | 2 | 1 | 0 |
| I Don't Know | 2 | 2 | 3 |

**Table 5.19:** *Results of question: "A Base Case Is A Subproblem With A Recursive Solution"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | False | True | I Don't Know |
| False | 8 | 2 | 1 |
| True | 3 | 1 | 3 |
| I Don't Know | 3 | 1 | 3 |

Sixteen of the participants answered the next question correctly on the post-survey. More students answered this question correctly than the previous question.

These results seem to indicate that students do not completely understand the role of base cases. Many students seem to believe that base cases guarantee termination of a recursive function.

The last three questions have no connection to the Recursive Breakout Game. The first of these questions concerned converting loops to equivalent recursive functions. This was discussed during the lesson regarding recursion. Fourteen of the participants answered this question correctly on the post-survey which is a modest improvement. The last two questions were included because binary trees were also discussed during the lesson about recursion. Every participant answered both binary tree questions correctly on the post-survey.

**Table 5.20:** *Results of question: "A Recursive Function With Base Cases Will Always Finish"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | False | True | I Don't Know |
| False | 6 | 1 | 2 |
| True | 0 | 4 | 1 |
| I Don't Know | 1 | 2 | 3 |

**Table 5.21:** *Results of question: "A Recursive Function Without A Base Case Will Never Finish"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | True | False | I Don't Know |
| True | 11 | 0 | 1 |
| False | 1 | 0 | 0 |
| I Don't Know | 4 | 0 | 3 |

## 5.7 Analysis

### 5.7.1 Analysis for Attitude Information Questions

A Wilcoxon signed rank test (with continuity correction) was performed on the results for each of the attitude questions [42]. This test was chosen because it is appropriate for discrete ordinal data such as Likert scale responses. The pre-survey and post-survey data was loaded into the R statistical system. The p-value for each question is shown in the table 5.25. Since $p<0.05$ is considered statistically significant, the results for all of the attitude questions except for "I Understand the Concept of Base Cases" are significant. Not all of the individual effects are positive: for nearly all of the questions one or two responses show a negative effect. Except for the last question, the number of positive changes is much greater than the number of negative effects.

**Table 5.22:** *Results of question: "Any For Loop Can Be Rewritten As An Equivalent Recursive Function"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | True | False | I Don't Know |
| True | 11 | 0 | 2 |
| False | 1 | 1 | 1 |
| I Don't Know | 2 | 0 | 2 |

**Table 5.23:** *Results of question: "Binary Trees Can Be Searched Recursively"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | True | False | I Don't Know |
| True | 19 | 0 | 0 |
| False | 0 | 0 | 0 |
| I Don't Know | 1 | 0 | 0 |

### 5.7.2 Analysis for Knowledge Questions

A Fisher's Exact Test, as described in the following section, for 3 by 3 contingency tables was performed on the responses to the knowledge questions. This test can easily be performed for contingency tables of modest sizes using the R System [62]. The p-value for each question is shown in Table 5.26.

The results for four of the questions are statistically significant ($p < 0.05$). The first of these, "A Recursive Function Combines The Solutions of Simpler Problems Into The Solution Of The Original Problem", is concerned with the overall purpose of a recursive function. The improvement is clear: 5 participants who were unsure during the pre-test answered correctly on the post-test. Because the overall number of participants is small this represents an improvement by 25% of the study participants.

**Table 5.24:** *Results of question: "Binary Trees Can Be Defined Recursively"*

| | Post-Survey | | |
|---|---|---|---|
| Pre-Survey | True | False | I Don't Know |
| True | 12 | 0 | 0 |
| False | 2 | 0 | 0 |
| I Don't Know | 6 | 0 | 0 |

**Table 5.25:** *Analysis of Attitude Questions*

| Question | p-value |
|---|---|
| I Understand the Concept of Recursion | 0.0056 |
| I Know What A Recursive Function Is | 0.0042 |
| Recursion is Difficult to Understand | 0.0308 |
| I Understand How Recursive Functions Are Called | 0.0058 |
| Recursion is Complex | 0.0164 |
| I Understand the Concept of Base Cases | 0.3198 |

The remaining questions concern the concept of base cases. One question, "A Base Case Is A Subproblem With A Non Recursive Solution", is included to test definitional knowledge of base cases. We also asked the corresponding contrary question, "A Base Case Is A Subproblem With A Recursive Solution". The results for this contrary question were not statistically significant.

The results for the next question, "A Recursive Function Cannot Have More Than One Base Case", show only a slight positive effect. There were seven correct responses on the pre-test. One additional participant answered correctly on the post-test. The results are statistically significant mostly because four people changed from "I Don't Know" to "True" (which is the incorrect response). This implies that the game appears to mislead many participants who are unsure into believing that there cannot be more than one base case for a recursive function.

For the remaining question, "A Recursive Function With Base Cases Will Always Finish", the results show a negative effect. Nine participants answered correctly on the pre-test and seven answered correctly on the post-test. It is interesting to note that one-third of those who responded correctly on the pre-test changed their responses on the post-test. Additionally half of the people who responded "I Don't Know" changed their answers on the post-test. One changed to the correct answer and two changed to the incorrect answer. It appears that the game caused participants who are unsure to change their responses and that it also misled many to change from correct responses to incorrect responses.

Clearly the p-values for the remaining questions are greater than 0.05. None of these results are considered statistically significant. Several of the data tables (in particular Tables 5.14 and 5.21) do show modest improvements however.

**Table 5.26:** *p-values of Knowledge Questions*

| Question | p-value |
|---|---|
| A Recursive Function Is A Function Which Calls Itself | 1 |
| A Recursive Function Cannot Call Other Functions | 1 |
| A Recursive Function Divides A Problem Into Simpler Problems | 0.0854 |
| A Recursive Function Combines The Solutions of Simpler Problems... | 0.0316 |
| A Recursive Function Must Have Exactly One Base Case | 0.1244 |
| A Base Case Is A Subproblem With A Non-Recursive Solution | 0.0454 |
| A Recursive Function Cannot Have More Than One Base Case | 0.0070 |
| A Base Case Is A Subproblem With A Recursive Solution | 0.4789 |
| A Recursive Function With Base Cases Will Always Finish | 0.0434 |
| A Recursive Function Must Return A Value | 0.1860 |
| Any For Loop Can Be Rewritten As An Equivalent Recursive Function | 0.0850 |
| A Recursive Function Without A Base Case Will Never Finish | 0.1986 |
| A Recursive Function Must Have At Least One Base Case | 0.3684 |
| Binary Trees Can Be Searched Recursively | 1 |
| Binary Trees Can Be Defined Recursively | 1 |

## 5.8 Discussion

One valid criticism is that because there was a discussion of recursion following gameplay it is not possible to determine whether the game itself is responsible for the changes observed. An additional carefully designed study is warranted to determine if the game by itself is effective. However it does not mean that the game is not useful for visualizing recursion or as an instructional aid. The results of the attitude questions bear this out. The majority of the students believed that the game should be used in the classroom.

It is also possible that players could play through the game successfully without necessarily learning anything about the concepts of recursion. Success in the game does not inherently require knowledge of recursion.

Another criticism is that the number of participants is rather small for sound statistical analysis. This is why an exact test was chosen. Exact tests are valid even when the number of participants is less than twenty. In spite of this, we believe further investigation with a larger participant population is warranted.

Clearly the first two of our hypotheses were confirmed. The results for both of the questions, "I Understand the Concept of Recursion" and "I Know What a Recursive Function Is" show positive and significant effects. Similarly, the students beliefs regarding the complexity and difficulty show positive and significant effects. Students also show greater confidence in knowing how recursive functions are called. These results indicate that even a short period of student engagement can have positive effect on student confidence and their beliefs about the complexity and difficulty of recursion.

The third hypothesis regarding improvement of general student knowledge of recursive functions was not confirmed. Students did not show significant improvement because the

majority answered the first survey question correctly on the pre-survey. One could argue that improvements shown in the other knowledge questions also show general improvement of knowledge but those other knowledge questions are covered by our remaining hypotheses.

The divide, conquer, and glue hypothesis was confirmed for the last two portions. Results for the question, "A Recursive Function Divides A Problem Into Simpler Problems" showed modest improvement which were not significant. The "conquer" part is arguably covered by the results for the question, "A Base Case Is a Subproblem With A Non-Recursive Solution". That question showed significant positive improvement in student knowledge. The "glue" question, "A Recursive Function Combines The Solutions of Simpler Problems Into The Solution Of The Original Problem" shows similar significant positive improvements. We believe that the game would be improved by a more explicit representation of the problem division portion of the execution. The current version only shows a model of the currently executing subproblem. Perhaps other subproblems at the same "level" of execution should be shown waiting to be executed.

Our fifth hypothesis concerned whether recursive functions can call other functions and whether they must return a value. This hypothesis was not confirmed. This is not a surprise since neither "other" functions nor function return values are explicitly represented in Recursive Breakout. We conjecture that adding individual non-recursive game levels might provide a solution regarding the first part of the hypothesis. These new game levels could be anything such as a level of Asteroids or a game of Solitaire. It is unclear how to represent return values within the game.

The sixth hypothesis concerns how many base cases are allowed or required. Participants were asked whether a recursive function must have at least one base case, whether a recursive function must have exactly one base case, and whether a recursive function cannot have more than one base case. Results were only statistically significant for the last of these questions. This question shows a slight negative effect: four participants who answered "I Don't Know" on the pre-survey changed their answer to the incorrect response on the post-survey. We believe that this may be caused by the fact that the game only has one type of base case (i.e. all base case game levels are identical). This may mislead participants into thinking that all recursive functions have only one base case. Perhaps the addition of different types of recursive bricks and base case levels would correct this.

The final hypothesis, "Playing Recursive Breakout will improve student knowledge of termination requirements for recursive functions", was not confirmed. This hypothesis was tested using two questions. The first question, "A Recursive Function With Base Cases Will Always Finish", had results which are statistically significant but the effect is unclear. The second question, "A Recursive Function Without A Base Case Will Never Finish", had results which are not statistically significant yet five additional participants answered correctly on the post-survey.

Of the attitude questions, "I Understand the Concept of Base Cases" is the only one which did not show significant result effects. As stated earlier, this indicates that the game does not represent base cases explicitly enough. This is clear from the results of the knowledge questions related to base cases as well. Perhaps in future versions, the game levels which represent base cases should be visually different so as to emphasize their place in the overall recursion.

## 5.9 Summary

We have shown that a simple game can significantly change student attitudes toward the concept of recursion. The game also appears to improve understanding of some elementary concepts, but for many concepts (e.g. base cases) the improvement is not statistically significant. It appears that students believe that they know more about recursion and are more confident in their understanding but the measured improvement in actual knowledge is less significant. The attitude of the participants toward recursion did change in a positive manner. Positively changing student attitude toward a concept is a first step in improving learning outcomes. We conjecture that the game had a positive effect both because of the illustrative effect from the visualization, and, because the participants enjoyed playing the game thus holding their attention longer and making them more receptive to learning.

The game should be considered an additional tool to assist in understanding recursion. Our game is a significant first step toward future games which can effectively teach all of the underlying concepts. One possible path for future work is to modify the game to better represent base cases and the divide portion of the divide, conquer, glue characterization of recursive functions.

# Chapter 6

# Conclusion

## 6.1 Contribution of the Research

We have demonstrated that it is possible to develop simple video games which teach fundamental concepts of Computer Science. We created two video games: one to teach boolean operators and one to teach the concept of recursion. To determine their effectiveness, we performed several quasi-experiments (there were no control groups in these studies) involving college and high school students.

For the boolean operator game, both college and high school students participated in our study. The students completed a pre-survey, played the game, and then completed a post-survey. The survey questions were too easy for the college students making it difficult to measure the effect of playing the game. For the High School Study, the game had only one significant effect. Our game reinforced the interpretation that "and" means disjunction which was not the intended effect of the game.

For the recursion game, only college students participated in our study. In this study, the students completed a pre-survey, played the game, participated in a short lecture, and then completed a post-survey. Our game had significant positive impact on student attitudes toward recursion. The game also had significant impact on student knowledge regarding the definition of a base case, that base cases do not guarantee function termination, and aspects of the divide, conquer, glue characterization of recursion. A majority of the students thought the game was "fun" and that it should be used in future courses. Based on these survey results, we have shown that our video game is effective at changing student attitudes toward recursion. We also show that even a simple game can have a positive effect on learning outcomes.

## 6.2 Implications of the Research

### 6.2.1 Implications for Computer Science Education

We believe this work is the first to present video games which teach Computer Science concepts rather than computer programming. Both of these types of video games are important new areas of research for Computer Science and Education. Video games should be used as supplements to traditional teaching of both concepts and computer programming.

### 6.2.2 Implications for Algorithm Visualization

Naps et. al. [127] identified different levels of user engagement for algorithm visualization systems. Playing a video game is not one of the engagement levels that were identified but our game appears to be an effective visualization of a recursive algorithm. Where does

"playing" belong in the characterization of user engagement? Is it equivalent to one of the other levels or is it something new? Also, will algorithm visualizations be more prevalent in classrooms if they are part of a game that can be "played"?

### 6.2.3 Implications for Future Research

Based on the success of the Recursive Breakout game, it would be worthwhile to try to improve it both as a game and as a teaching tool. Possible improvements to the game include the use of color, more types of bricks, and multi-ball play. To improve as a teaching tool, the game should have a better representation of base cases.

One obvious area of research would be to design and build games which teach other basic concepts such as iteration, functions, and variables. Games could also be created for teaching concepts of object-oriented programming. We would also revisit the idea of a game for teaching boolean operations.

Another area of research would be to design and build "playable visualizations" of algorithms. This work would be informed by the existing work in algorithm visualization. We envision games which are highly configurable so that instructors or the game itself could provide levels and situations tailored to an individual student's progress and learning needs.

Finally, it would be interesting to design video games which can automatically and inherently determine if it is effectively assisting the student. This ability would avoid having to conduct separate surveys and may also allow long term monitoring of student progress.

## 6.3 Limitations of Study

The boolean game had little effect on the learning outcomes of the high school students. Part of this may be due to student attitudes or that the interaction time with the game was too short. Another study should be conducted.

The study of the boolean game with the college students was conducted more than half way through a CS1 course. This was too late, as the students had already been exposed to boolean expressions and conditional statements. The pre-survey questions were too easy and quite a number of students correctly answered all of them. This made it difficult to measure the effect of the game on student knowledge. Another study should be conducted earlier in the course to see if the boolean game is effective with college level students.

The study of the recursion game had only 20 participants. All of the participants were enrolled in a CS1 course. A larger study should be conducted to determine if the results still hold for larger and more diverse populations.

## 6.4 Final Conclusion

In this dissertation, we have shown that video games can be designed to teach computer science concepts. To our knowledge these are the first video games intended to teach Computer Science concepts rather than computer programming. Even though our games only had a positive effect for a small number of survey questions, these results are sufficiently positive to call for more research in this and other related areas. Our work presented here is only a small first step toward using video games to teach fundamental Computer Science concepts.

# Bibliography


[1] The alice programming environment. www.alice.org.

[2] America's army. www.americasarmy.com.

[3] Asteroids. www.atari.com.

[4] Bureau of labor and statistics. www.bls.gov/oco/ocos303.htm.

[5] Centipede. www.atari.com.

[6] Cheapass games. www.cheapass.com.

[7] Clue. www.hasbro.com.

[8] Darfur is dying. www.darfurisdying.com.

[9] The education arcade. www.educationarcade.org.

[10] Every extend. nagoya.cool.ne.jp/omega.

[11] Game maker. www.yoyogames.com/gamemaker.

[12] Games for change. www.gamesforchange.org.

[13] Games for health. www.gamesforhealth.org.

[14] Gearbox software. www.gearboxsoftware.com.

[15] Gonzalo frasca. en.wikipedia.orgwikiGonzaloFrasca.

[16] The greenfoot programming environment. www.greenfoot.org.

[17] The howard dean for america game. www.deanforamericagame.com.

[18] Ian bogost. www.bogost.com.

[19] Institute for creative technology. ict.usc.edu.

[20] Jflap: Formal language visualization system. www.jflap.org.

[21] Jhave: Algorithm visualization. jhave.org.

[22] Lightbot game. www.armorgames.com.

[23] The magic school bus. www.scholastic.com.

[24] Math blaster. www.knowledgeadventure.com.

[25] Missile command. www.atari.com.

[26] Muzzy lane. www.muzzylane.com.

[27] Oregon trail. www.riverdeep.net.

[28] P4games project. www.p4games.org.

[29] Pez candy. www.pez.com.

[30] Political machine. www.politicalmachine.com.

[31] Rafael fajardo. www.rafaelfajardo.com.

[32] Reader rabbit. www.reader-rabbit.com.

[33] Robot odyssey. en.wikipedia.org/wiki/RobotOdyssey.

[34] Rocky's boots. www.warrenrobinett.com/rockysboots/.

[35] Serious games. www.seriousgames.org.

[36] Under ash. www.underash.net.

[37] Video game chartz. www.vgchartz.com.

[38] Where is the world is carmen sandiego? www.broderbund.com.

[39] Zero hour. www.nemspi.org.

[40] Frank Ackerman, Tim McGuire, Terry Scott, and John Peterson. Nifty assignments. *J. Comput. Small Coll.*, 24(1):257–262, 2008.

[41] Nicoletta Adamo-Villani and Ronnie Wilbur. An immersive game for k-5 math and science education. *11th International Conference Information Visualization (IV'07)*, 2007.

[42] Alan Agresti. *Categorical Data Analysis*. John Wiley and Sons, 1990.

[43] Go..knur Kaplan Akilli. *Games and Simulations in Online Learning: Research and Development Frameworks*, chapter Games and Simulations: A New Approach in Education?, pages 1–20. Information Science Publishing, 2007.

[44] Mohammed Al-Bow, Debra Austin, Jeffrey Edgington, Rafael Fajardo, Joshua Fishburn, Carlos Lara, Scott Leutenegger, and Susan Meyer. Using greenfoot and games to teach rising 9th and 10th grade novice programmers. In *Sandbox '08: Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, pages 55–59, New York, NY, USA, 2008. ACM.

[45] Mohammed Al-Bow, Debra Austin, Jeffrey Edgington, Rafael Fajardo, Joshua Fishburn, Carlos Lara, Scott Leutenegger, and Susan Meyer. Using game creation for teaching computer programming to high school students and teachers. In *ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education*, pages 104–108, New York, NY, USA, 2009. ACM.

[46] Vicki L. Almstrum, Peter B. Henderson, Valerie Harvey, Cinda Heeren, William Marion, Charles Riedesel, Leen-Kiat Soh, and Allison Elliot Tew. Concept inventories in computer science for the topic discrete mathematics. In *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, pages 132–145, 2006.

[47] J. Ángel Velázquez-Iturbide. Recursion in gradual steps (is recursion really that difficult?). *SIGCSE Bull.*, 32(1):310–314, 2000.

[48] Lawrence Argent, Bill Depper, Rafael Fajardo, Sarah Gjertson, Scott T. Leutenegger, Mario A. Lopez, and Jeff Rutenbeck. Building a game development program. *Computer*, 39:52–60, 2006.

[49] Tiffany Barnes, Eve Powell, Amanda Chaffin, and Heather Lipford. Game2learn: improving the motivation of cs1 students. In *GDCSE '08: Proceedings of the 3rd international conference on Game development in computer science education*, pages 1–5, New York, NY, USA, 2008. ACM.

[50] Katrin Becker. *Games and Simulations in Online Learning: Research and Development Frameworks*, chapter Pedagogy in Commercial Video Games, pages 21–47. Information Science Publishing, 2007.

[51] Katrin Becker and J. R. Parker. Serious games + computer science = serious cs. *J. Comput. Small Coll.*, 23(2):40–46, 2007.

[52] Mordechai Ben-Ari. Recursion: From drama to program. In *Aspects of Teaching Computer Science*, volume 7, pages 45–47, 1996.

[53] Kim B. Bruce, Andrea Danyluk, and Thomas Murtagh. Why structural recursion should be taught before arrays in cs 1. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 246–250, New York, NY, USA, 2005. ACM.

[54] M. D. Byrne, R. Catrambone, and J. T. Stasko. Evaluating animations as student aids in learning computer algorithms. *Computer and Education*, 33:253–278, 1999.

[55] Ruth M. J. Byrne and P. N. Johnson-Laird. 'if' and the problems of conditional reasoning. *Trends in Cognitive Sciences*, 13(7):282–286, 2009.

[56] Amanda Chaffin, Katelyn Doran, Drew Hicks, and Tiffany Barnes. Experimental evaluation of teaching recursion in a video game. In *Sandbox '09: Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, pages 79–86, New York, NY, USA, 2009. ACM.

[57] D. Cliburn. The effectiveness of games as assignments in an introductory programming course. In *Proceedings of the 36th Annual ASEE/IEEE Frontiers in Education Conference (FIE 2006)*, 2006.

[58] Daniel C. Cliburn and Susan Miller. Games, stories, or something more traditional: the types of assignments college students prefer. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 138–142, New York, NY, USA, 2008. ACM.

[59] Daniel C. Cliburn and Susan M. Miller. What makes a "good" game programming assignment? *J. Comput. Small Coll.*, 23(4):201–207, 2008.

[60] M. Cole and S. R. Cole. *The development of children*. W. H. Freeman, 1996.

[61] Michelle Craig and Diane Horton. Gr8 designs for gr8 girls: a middle-school program and its evaluation. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 221–225, New York, NY, USA, 2009. ACM.

[62] Michael J. Crawley. *The R Book*. John Wiley and Sons, 2007.

[63] Wanda Dann, Stephen Cooper, and Randy Pausch. Using visualization to teach novices recursion. In *ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 109–112, New York, NY, USA, 2001. ACM.

[64] Michael Eagle and Tiffany Barnes. Experimental evaluation of an educational game for improved learning in introductory computing. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 321–325, New York, NY, USA, 2009. ACM.

[65] Richard Van Eck. *Games and Simulations in Online Learning: Research and Development Frameworks*, chapter Building Artificially Intelligent Learning Games, pages 271–307. Information Science Publishing, 2007.

[66] Jeffrey Edgington. Teaching and viewing recursion as delegation. *J. Comput. Small Coll.*, 23(1):241–246, 2007.

[67] Jeffrey Edgington and Scott Leutenegger. Using the ancient game of rogue in cs1. *J. Comput. Small Coll.*, 24(1):150–156, 2008.

[68] M. Falco. Set: The family game of visual perception. www.setgame.com.

[69] Matthew Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. *How To Design Programs*. MIT Press, 2000.

[70] Shalom M. Fisch. Making educational computer games "educational". In *IDC '05: Proceedings of the 2005 conference on Interaction design and children*, pages 56–61, New York, NY, USA, 2005. ACM.

[71] Gary Ford. A framework for teaching recursion. *SIGCSE Bull.*, 14(2):32–39, 1982.

[72] Gary Ford. An implementation-independent approach to teaching recursion. *ACM*, 1984.

[73] Herve Franceschi. Use of animation director movies to teach cs1 programming concepts. *J. Comput. Small Coll.*, 20(3):19–27, 2005.

[74] Tracy Fullerton. Play-centric games education. *Computer*, 39(6):36–42, 2006.

[75] Gagne. *Conditions of Learning*. 1965.

[76] J. P. Gee. *What video games have to teach us about learning and literacy*. Palgrave-MacMillan, 2003.

[77] J. P. Gee. *Good video games + good learning: collected essays on video games, learning and literacy*. Peter Lang Publishing, 2007.

[78] Carlisle E. George. Erosi—visualising recursion and discovering new errors. In *SIGCSE '00: Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, pages 305–309, New York, NY, USA, 2000. ACM.

[79] Paul Gestwicki and Fu-Shing Sun. Teaching design patterns through computer game development. *J. Educ. Resour. Comput.*, 8(1):1–22, 2008.

[80] Paul V. Gestwicki. Computer games as motivation for design patterns. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 233–237, New York, NY, USA, 2007. ACM.

[81] Ray Giguette. The crawfish and the aztec treasure maze: adventures in data structures. *SIGCSE Bull.*, 34(4):89–93, 2002.

[82] Ray Giguette. Pre-games: games designed to introduce cs1 and cs2 programming assignments. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 288–292, New York, NY, USA, 2003. ACM.

[83] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. Identifying important and difficult concepts in introductory computing courses using a delphi process. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 256–260, New York, NY, USA, 2008. ACM.

[84] C. M. Gorriz and C. Medina. Engaging girls with computers through software games. *Communications of the ACM*, 43(1):42–49, 2000.

[85] Tina Götschi, Ian Sanders, and Vashti Galpin. Mental models of recursion. In *SIGCSE '03: Proceedings of the 34rd SIGCSE technical symposium on Computer science education*, pages 346–350, 2003.

[86] Sharon L. Greene, Susan J. Devlin, Philip E. Cannata, and Louis M. Gomez. No ifs, ands, or ors: A study of database querying. *International Journal of Man-Machine Studies*, 32(3):303 – 326, 1990.

[87] P. M. Greenfield, P. deWinstanley, H. Kilpatrick, and D. Kaye. Action video games and informal education: Effects on strategies for dividing visual attention. *Journal of Applied Developmental Psychology*, 15(1):105–123, 1994.

[88] Scott Grissom, Myles F. McNally, and Tom Naps. Algorithm visualization in cs education: comparing levels of student engagement. In *SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization*, pages 87–94, New York, NY, USA, 2003. ACM.

[89] Katherine Gunion, Todd Milford, and Ulrike Stege. Curing recursion aversion. In *ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education*, pages 124–128, New York, NY, USA, 2009. ACM.

[90] Bruria Haberman and Haim Averbuch. The case of base cases: why are they so difficult to recognize? student difficulties with recursion. *SIGCSE Bull.*, 34(3):84–88, 2002.

[91] Patricia Haden. The incredible rainbow spitting chicken: teaching traditional programming skills through games programming. In *ACE '06: Proceedings of the 8th Austalian conference on Computing education*, pages 81–89, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[92] Brian Hanks. Problems encountered by novice pair programmers. *ACM Journal of Educational Resources in Computing*, 7(4), 2008.

[93] Stuart Hansen. The game of set®: an ideal example for introducing polymorphism and design patterns. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 110–114, New York, NY, USA, 2004. ACM.

[94] B. Harvey. *Computer Science Logo Style Volume 1: Symbolic Computing 2/e*. MIT Press, 1997.

[95] S. M. Haynes. Explaining recursion to the unsophisticated. *SIGCSE Bull.*, 27(3), 1995.

[96] J.-M Hoc. *Studying the Novice Programmer*, chapter Do We Really Have Conditional Statements in Our Brains?, pages 179–190. Lawrence Erlbaum Associates, 1989.

[97] Ian Horswill and Marlena Novak. Evolving the artist-technologist. *Computer*, 39(6):62–69, 2006.

[98] Timothy Huang. Strategy game programming projects. *Journal of Computing in Small Colleges*, 16(4), May 2001.

[99] Timothy Huang. The game of go: An ideal environment for capstone and undergraduate research projects. In *SIGCSE '03: Proceedings of the 34rd SIGCSE technical symposium on Computer science education*, 2003.

[100] Christopher D. Hundhausen, S. A. Douglas, and J. T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13(3):259–290, 2002.

[101] Jiwen Huo and Wm Cowan. Comprehending boolean queries. In *APGV '08: Proceedings of the 5th symposium on Applied perception in graphics and visualization*, pages 179–186, New York, NY, USA, 2008. ACM.

[102] Petri Ihantola, Ville Karavirta, Ari Korhonen, and Jussi Nikander. Taxonomy of effortless creation of algorithm visualizations. In *ICER '05: Proceedings of the first international workshop on Computing education research*, pages 123–133, New York, NY, USA, 2005. ACM.

[103] D. J. Jarc, M. B. Feldman, and R. S. Heller. Accessing the benefits of interactive prediction using web-based algorithm animation courseware. In *Proceedings of the SIGCSE Session*, 2000.

[104] P. N. Johnson-Laird. *Mental Models*. Cambridge University Press, 1983.

[105] P. N. Johnson-Laird. The history of mental models. Technical report, 2005.

[106] Hank Kahney. *Studying the Novice Programmer*, chapter What do novice programmers know about recursion?, pages 209–228. Lawrence Erlbaum Associates, 1989.

[107] Jennifer S. Kay. Using the force: how star wars can help you teach recursion. In *CCSC '00: Proceedings of the fifth annual CCSC northeastern conference on The journal of computing in small colleges*, pages 274–284, , USA, 2000. Consortium for Computing Sciences in Colleges.

[108] Fengfeng Ke. A case study of computer gaming for math: Engaged learning from gameplay? *Computers and Education*, 51:1609–1620, 2008.

[109] Claudius M. Kessler and John R. Anderson. Learning flow of control: Recursive and iterative procedures. *Human-Computer Interaction*, 2:135–166, 1986.

[110] Eric Klopfer, Scot Osterweil, and Katie Salen. Moving learning games forward. Technical report, The Education Arcade, 2009.

[111] Michael Kolling and Poul Henriksen. Game programming in introductory courses with direct state manipulation. In *ITICSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, June 2005.

[112] Robert L. Kruse. On teaching recursion. *ACM*, 1982.

[113] D. Midian Kurland and Roy D. Pea. Children's mental models of recursive logo programs. *Journal of Educational Computing Research*, 1(2):235–243, 1985.

[114] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. In *ITICSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 14–18, June 2005.

[115] Brenda K. Laurel. chapter Interface as Mimesis. 1986.

[116] R. Lawrence. Teaching data structures using competitive games. In *IEEE Transactions on Education*, volume 47, 2004.

[117] Scott Leutenegger and Jeffrey Edgington. A games first approach to teaching introductory programming. *SIGCSE Bull.*, 39(1):115–118, 2007.

[118] Anany Levitin and Mary-Angela Papalaskari. Using puzzles in teaching algorithms. In *SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, 2002.

[119] Geoffrey R. Loftus and Walter Nelson. Games as teaching tools: The computer connection. *Computers in the Schools*, 1(4):67–76, April 1985.

[120] David J. Malan and Henry H. Leitner. Scratch for budding computer scientists. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 223–227, New York, NY, USA, 2007. ACM.

[121] Linda Kathryn McIver. *Syntactic and Semantic Issues in Introductory Programming Education*. PhD thesis, Monash University, January 2001.

[122] Myles McNally, Thomas Naps, Scott Grissom, David Furcy, and Christian Trefftz. Supporting the rapid development of pedagogically effective algorithm visualizations. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, 2007.

[123] Mecc. Oregon trail. 1974.

[124] A. Michard. Graphical presentation of boolean expressions in a database query language: design notes and an ergonomic evaluation. *Behaviour and Information Technology*, 1(3):279–288, 1982.

[125] Janet Murray, Ian Bogost, Michael Mateas, and Michael Nitsche. Game design education: Integrating computation and culture. *Computer*, 39(6):43–51, 2006.

[126] Brad A. Myers, Andrew J. Ko, and Margaret M. Burnett. Invited research overview: end-user programming. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 75–80, New York, NY, USA, 2006. ACM.

[127] Thomas Naps, Stephen Cooper, Boris Koldehofe, Charles Leska, Guido Rossling, Wanda Dann, Ari Korhonen, Lauri Malmi, Jarmo Rantakokko, Rockford J. Ross, Jay Anderson, Rudolf Fleischer, Marja Kuittinen, and Myles McNally. Evaluating the educational impact of visualization. In *ITiCSE-WGR '03: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 124–136, New York, NY, USA, 2003. ACM.

[128] John F. Pane. *A Programming System for Children that is Designed for Usability*. PhD thesis, Carnegie Mellon University, 2002.

[129] John F. Pane and Brad A. Myers. Natural and accurate ways to specify the selection of objects from a group. 1900.

[130] John F. Pane and Brad A. Myers. Tabular and textual methods for selecting objects from a group. In *Proceedings of the 2000 IEEE International Symposium on Visual Languages (VL'00)*, 2000.

[131] Nick Parlante. Nifty assignments. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 112–113, New York, NY, USA, 2008. ACM.

[132] Nick Parlante, John Cigas, Angela B. Shiflet, Raja Sooriamurthi, Mike Clancy, Bob Noonan, and David Reed. Nifty assignments. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 497–498, New York, NY, USA, 2007. ACM.

[133] Nick Parlante, David Levine, Steven Andrianoff, Aaron J. Gordon, Alyce Brady, Pamela Cutter, Paul Kube, Jefferson Ng, and Richard E. Pattis. Nifty assignment. *SIGCSE Bull.*, 37(1):371–372, 2005.

[134] Nick Parlante, David Matuszek, Jeff Lehman, David Reed, John K. Estell, and Donald Chinn. Nifty assignments. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 46–47, New York, NY, USA, 2004. ACM.

[135] Nick Parlante, Steven A. Wolfman, Lester I. McCann, Eric Roberts, Chris Nevison, John Motil, Jerry Cain, and Stuart Reges. Nifty assignments. *SIGCSE Bull.*, 38(1):562–563, 2006.

[136] Lakshmi Prayaga. Game programming - the "why", "what" and "how" with graphics objects. *Journal of Object Technology*, 4(9):39–58, 2005.

[137] Lakshmi Prayaga. Game technology as a tool to actively engage k-12 students in the act of learning. In *SIGITE'05*, 2005.

[138] Marc Prensky. *Digital game-based learning*. McGraw-Hill, 2001.

[139] Blaine A. Price, Ronald M. Baecher, and Ian S. Small. A principled taxonomy of software visualization. *Journal of Visual Languages and Computing*, 4(3):211–266, 1993.

[140] R. V. Price. *Computer-aided instruction: A guide for authors*. BrooksCole Publishing Company, 1990.

[141] E. F. Provenzo. The video generation. *The American School Board Journal*, 179(3):29–32, 1992.

[142] Yolanda Rankin, Amy Gooch, and Bruce Gooch. The impact of game design on students' interest in cs. In *GDCSE '08: Proceedings of the 3rd international conference on Game development in computer science education*, pages 31–35, New York, NY, USA, 2008. ACM.

[143] Richard Rassala, Jeff Raab, and Viera K. Proulx. The sigcse 2001 maze demostration program. In *SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, 2002.

[144] L. P. Rieber. Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games. *Educational Technology Research and Development*, 44(2):43–58, 1996.

[145] Guido Rössling, Thomas Naps, Mark S. Hall, Ville Karavirta, Andreas Kerren, Charles Leska, Andrés Moreno, Rainer Oechsle, Susan H. Rodger, Jaime Urquiza-Fuentes, and J. Ángel Velázquez-Iturbide. Merging interactive visualizations with hypertextbooks and course management. In *ITiCSE-WGR '06: Working group reports on ITiCSE on Innovation and technology in computer science education*, pages 166–181, New York, NY, USA, 2006. ACM.

[146] Katie Salen and Eric Zimmerman. *Rules of Play*. MIT Press, 2003.

[147] Ian Sanders, Vashti Galpin, and Tina Götschi. Mental models of recursion revisited. In *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, pages 138–142, New York, NY, USA, 2006. ACM.

[148] Carsten Schulte and Jens Bennedsen. What do teachers teach in introductory programming? In *ICER '06: Proceedings of the second international workshop on Computing education research*, pages 17–28, New York, NY, USA, 2006. ACM.

[149] R. Sedgewick. *Algorithms*. Addison-Wesley, 1988.

[150] Clifford A. Shaffer, Matthew Cooper, and Stephen H. Edwards. Algorithm visualization: a report on the state of the field. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 150–154, New York, NY, USA, 2007. ACM.

[151] Ben Shneiderman. *Designing the User Interface - Strategies for Effective Human-Computer Interaction*. Addison-Wesley, second edition, 1992.

[152] Kurt Squire, Mike Barnett, Jamillah M. Grant, and Thomas Higginbotham. Electromagnetism supercharged!: learning physics with digital simulation games. In *ICLS '04: Proceedings of the 6th international conference on Learning sciences*, pages 513–520. International Society of the Learning Sciences, 2004.

[153] J. Stasko. *Software Visualization*. 1998.

[154] Ben Stephenson. Using graphical examples to motivate the study of recursion. *J. Comput. Small Coll.*, 25(1):42–50, 2009.

[155] K. Subrahmanyam, P. Greenfield, R. Kraut, and E. Gross. The impact of computer use on children's and adolescents' development. *Applied Developmental Psychology*, 22(1):7–30, 2001.

[156] David M. Temple and Steve P. Guest. *Cognitive Aspects of Visual Languages and Visual Interfaces*, chapter Diagrammatic Techniques for the Visualisation of Object Oriented Programming, pages 259–294. Elsevier Science, 1994.

[157] Sho-Huan Tung, Ching-Tao Chang, Wing-Kwong Wong, and Jihn-Chang Jehng. Visual representations for recursion. *Int. J. Hum.-Comput. Stud.*, 54(3):285–300, 2001.

[158] Franklyn Turbak, Constance Royden, Jennifer Stephan, and Jean Herbst. Teaching recursion before loops in cs1. *Journal of Computing in Small Colleges*, 14(4):86–101, 1999.

[159] Gerrit C. van der Veer. *Cognitive Aspects of Visual Languages and Visual Interfaces*, chapter Mental Models of Computer Systems: Visual Languages in the Mind, pages 3–40. Elsevier Science, 1994.

[160] Scott A. Wallace and Jason Margolis. Exploring the use of competetive programming: observations from the classroom. *J. Comput. Small Coll.*, 23(2):33–39, 2007.

[161] Linda L. Werner, Shannon Campe, and Jill Denner. Middle school girls + games programming = information technology fluency. In *SIGITE'05*, 2005.

[162] Cheng-Chih Wu, Nell B. Dale, and Lowell J. Bethel. Conceptual models and cognitive learning styles in teaching recursion. *SIGCSE Bull.*, 30(1):292–296, 1998.

[163] Michael Zyda. From visual simulation to virtual reality to games. *IEEE Computer*, 2005.

[164] Michael Zyda, Dhruv Thukral, Sumeet Jakatdar, Jonathan Engelsma, James Ferrans, Mat Hans, Larry Shi, Fred Kitson, and Venu Vasudevan. Educating the next generation of mobile game developers. *IEEE Comput. Graph. Appl.*, 27(2):96–95, 2007.

# Appendix A

# Exact Statistical Tests

Exact statistical tests were designed, in part, to calculate inferential statistics for small populations of participants and were initially created for use with 2 by 2 contingency tables. The specific test used to analyze our data was first described by R. A. Fisher in 1935. For 2 by 2 contingency tables, the exact distribution is based on "the marginal frequencies in both margins." [42]. The margins are the row and column sums. According to Agresti, if we assume that these values are fixed, then the distribution is hypergeometric:

$$\frac{\binom{n_{1+}}{n_{11}}\binom{n_{2+}}{n_{+1}-n_{11}}}{\binom{n}{n_{+1}}} \tag{A.1}$$

Note that the distribution only depends on the value in first cell of the contingency table (because we know the row and column sums, once we "choose" the value of the first cell the remaining values in the table can be calculated). We are interested in determining if the post-results are independent of the pre-results. The hypothesis and alternate hypothesis are based on the value of the odds ratio. The hypothesis is that the results are independent

(there is no association). The alternate hypothesis is that the results are associated in some way. The odds ratio, $\theta$, is defined (for 2 by 2 tables) as:

$$\theta = \frac{n_{11}n_{22}}{n_{12}n_{21}} \tag{A.2}$$

If the odds ratio, $\theta$, is 1 and all cell values are greater than zero, then the post-results are independent of the pre-values. If $\theta > 1$ then the two sets of results are not independent and larger values of $\theta$ "represent stronger levels of association" [42].

A table with a larger value in the first cell and the same row and column sums will produce a larger odds ratio than that of the observed table. Any such table supports the alternate hypothesis more than the observed table because it represents a set of results which has a greater association between pre- and post-survey results. The p-value we are interested in is the "hypergeometric probability that $n_{11}$ is at least as large as the observed value" [42]. We therefore examine this probability for each possible table with smaller values of $n_{11}$ and the same row and column sums. The p-value of the observed table is calculated by summing the p-values for all of these tables. The reasoning and calculation is similar for contingency tables with more columns and rows.

# Appendix B

# Boolean Game College Study Survey

All personal information gathered is private. Other information will only be used by the department and faculty after all personal information is removed (i.e. if information is used it will be anonymous). This survey has no effect on your grade in this course.

General Information

Name:

Age:

Gender: Female Male

Class Rank: Freshman Sophomore Junior Senior

What type of degree? Bachelor Arts Bachelor Science Other

What is your Major?

What is your Minor?

## Boolean Knowledge

| | True | False | Don't Know |
|---|---|---|---|
| A boolean variable can only store the values of true or false | | | |
| If statements allow a program to make decisions based on current conditions | | | |
| In Java, ! represents "not" | | | |
| In Java, ! represents "or" | | | |
| In Java, ! represents "and" | | | |
| In Java, ll represents "not" | | | |
| In Java, ll represents "or" | | | |
| In Java, ll represents "and" | | | |
| In Java, && represents "not" | | | |
| In Java, && represents "or" | | | |
| In Java, && represents "and" | | | |
| If ! X is true then X must be false | | | |
| If ! X is true then X must be true | | | |
| if ! X is false then X must be false | | | |
| If ! X is false then X must be true | | | |
| If X ll Y ll Z  is true, each of X, Y, Z must be true | | | |
| If X ll Y ll Z is true, any of X, Y, Z must be true | | | |
| If X ll Y ll Z is false, each of X, Y, Z must be false | | | |
| If X ll Y ll Z is false, any of X, Y, Z must be false | | | |
| If X && Y && Z is true, each of X, Y, Z must be true | | | |
| If X && Y && Z is true, any of X, Y, Z must be true | | | |
| If X && Y && Z is false, each of X, Y, Z must be false | | | |
| If X && Y && Z is false, any of X, Y, Z must be false | | | |

# Appendix C

# Boolean Game High School Study Survey

All personal information gathered is private. Other information will only be used by the department and faculty after all personal information is removed (i.e. if information is used it will be anonymous). This survey has no effect on your grade in this course.

Name:

**Choose all the objects that match blue**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match blue and circle**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match square or green**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match blue and green**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match red or match circle**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match blue and the objects that match circle**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match red, if the objects match triangle**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match blue, unless the objects match square**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match not red and square**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match square and not red**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Unless the objects match green, Choose all the objects that match circle**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match not (triangle or red)**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match (not circle) or blue**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match not triangle or green**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**Choose all the objects that match not (triangle and red)**

- Red Triangle
- Red Circle
- Red Square
- Blue Triangle
- Blue Circle
- Blue Square
- Green Triangle
- Green Circle
- Green Square

**If not X is true then X must be**

- False
- True
- I don’t know

**If not X is false then X must be**

- False
- True
- I don’t know

**If X or Y or Z is true then**

- At least one of X, Y, Z must be false
- At least one of X, Y, Z must be true
- Each of X, Y, Z must be false
- Each of X, Y, Z must be true
- I don’t know

**If X or Y or Z is false then**

- At least one of X, Y, Z must be false
- At least one of X, Y, Z must be true
- Each of X, Y, Z must be false
- Each of X, Y, Z must be true
- I don’t know

**If X and Y and Z is true then**

- At least one of X, Y, Z must be false
- At least one of X, Y, Z must be true
- Each of X, Y, Z must be false
- Each of X, Y, Z must be true
- I don’t know

**If X and Y and Z is false then**

- At least one of X, Y, Z must be false
- At least one of X, Y, Z must be true
- Each of X, Y, Z must be false
- Each of X, Y, Z must be true
- I don’t know

# Appendix D

# Recursion Game Pre-Survey

All personal information gathered is private. Other information will only be used by the department and faculty after all personal information is removed (i.e. if information is used it will be anonymous). This survey has no effect on your grade in this course.

General Information

Name:

Age:

Gender: Female Male

Class Rank: Freshman Sophomore Junior Senior

What type of degree? Bachelor Arts Bachelor Science Other

What is your major?

| | Strongly Agree | Agree | Neutral | Disagree | Strongly Disagree |
|---|---|---|---|---|---|
| I understand the concept of Recursion | | | | | |
| I know what a Recursive Function is | | | | | |
| Recursion is difficult to understand | | | | | |
| I understand how Recursive Functions are called | | | | | |
| Recursion is complex | | | | | |
| I understand the concept of Base Cases | | | | | |

**Recursion Knowledge**

| | True | False | Don't Know |
|---|---|---|---|
| A Recursive function is a function which calls itself | | | |
| A Recursive function cannot call other functions | | | |
| A Recursive function divides a problem into simpler problems | | | |
| A Recursive function combines the solutions of simpler problems into the solution of the initial problem | | | |
| A Recursive function must have exactly one base case | | | |
| A Base case is a subproblem with a non-recursive solution | | | |
| A Recursive function cannot have more than one base case | | | |
| A Base case is a subproblem with a recursive solution | | | |
| A Recursive function with base cases will always finish | | | |
| A Recursive function must return a value | | | |
| Any for loop can be rewritten as an equivalent recursive function | | | |
| A Recursive function without a base case will never finish | | | |
| A Recursive function must have at least one base case | | | |
| Binary trees can be searched recursively | | | |
| Binary trees can be defined recursively | | | |

# Appendix E

# Recursion Game Post-Survey

All personal information gathered is private. Other information will only be used by the department and faculty after all personal information is removed (i.e. if information is used it will be anonymous). This survey has no effect on your grade in this course.

General Information

Name:

Age:

Gender: Female Male

Class Rank: Freshman Sophomore Junior Senior

What type of degree? Bachelor Arts Bachelor Science Other

What is your major?

| | Strongly Agree | Agree | Neutral | Disagree | Strongly Disagree |
|---|---|---|---|---|---|
| I understand the concept of Recursion | | | | | |
| I know what a Recursive Function is | | | | | |
| Recursion is difficult to understand | | | | | |
| I understand how Recursive Functions are called | | | | | |
| Recursion is complex | | | | | |
| I understand the concept of Base Cases | | | | | |
| Playing the Recursive Breakout Game helped me understand Recursion | | | | | |
| The Recursive Breakout Game is fun | | | | | |
| The Recursive Breakout Game should be used in courses to teach Recursion | | | | | |

**Recursion Knowledge**

| | True | False | Don't Know |
|---|---|---|---|
| A Recursive function is a function which calls itself | | | |
| A Recursive function cannot call other functions | | | |
| A Recursive function divides a problem into simpler problems | | | |
| A Recursive function combines the solutions of simpler problems into the solution of the initial problem | | | |
| A Recursive function must have exactly one base case | | | |
| A Base case is a subproblem with a non-recursive solution | | | |
| A Recursive function cannot have more than one base case | | | |
| A Base case is a subproblem with a recursive solution | | | |
| A Recursive function with base cases will always finish | | | |
| A Recursive function must return a value | | | |
| Any for loop can be rewritten as an equivalent recursive function | | | |
| A Recursive function without a base case will never finish | | | |
| A Recursive function must have at least one base case | | | |
| Binary trees can be searched recursively | | | |
| Binary trees can be defined recursively | | | |