

GAME2LEARN: BUILDING A COMPILER INTO A GAME ENGINE TO
INCREASE LEARNING GAINS IN COMPUTER SCIENCE STUDENTS

by

Amanda Chaffin

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Computer Science

Charlotte

2009

Approved by:

Dr. Tiffany Barnes

Dr. G. Michael Youngblood

Dr. Jamie Payton

©2009
Amanda Brook Chaffin
ALL RIGHTS RESERVED

ABSTRACT

AMANDA BROOK CHAFFIN. Game2learn: building a compiler into a game engine to increase learning gains in computer science students. (Under the direction of DR. TIFFANY BARNES)

Increasingly, games are being developed for use inside the classroom and work environments as a learning tool for students and employees in a variety of disciplines. Since its inception in 2005, the Game2Learn group has been dedicated to the idea that games can be used to teach introductory computer science concepts to introductory students. The games developed by the group have always been driven by the need to increase student motivation and engagement in the computer science class, which is critical to gaining and retaining computer science students. The games built prior to this thesis were all rapid prototype games which, while showing learning gains and improved attitudes toward educational games for computing, did not allow the students to write actual code in the game environment, depending instead of code metaphors such as a magic code fairy that students input for loop numbers into the program (Wu's Castle), a drag and drop pseudo code interface (Saving Sera), and instruction through dialogue trees (StormHaven 2 and The Tournament). The hypothesis that we are targeting is that *EleMental: The Recurrence* will improve student learning about recursion and depth-first search. We also hypothesize that students who play the game will have a positive attitude toward playing such games for homework as opposed to traditional assignments. In this thesis, we will discuss the computer science concept that we targeted with our game design to meet our hypothesis, the game that we created, how it was implemented, and

present the results from the study. The study scores from the pretest to the posttest (with a $p < .05$) show learning gains from playing the game that we built and we also present the qualitative feedback from our participants, most of it highly favorable. Unlike our other Game2Learn papers, this thesis is dedicated to the idea that having students write actual code (in this case, C# code) which they can compile and run in the game world would be the ideal interface to teach students introductory computer science course.

ACKNOWLEDGEMENTS

It would not have been possible for this thesis to be completed without a number of individual's help and support during the entire process. First, I have to thank Dr. Tiffany Barnes who saw something in me, back in my undergraduate degree, that inspired her to select me for the Game2Learn project and then to be my thesis advisor. Without her, I would not have even applied for Graduate school, let alone completed it, and I have to thank her for the incredible journey this opportunity has been. I also owe a deep thank you to Dr. Michael Youngblood who, through the Game Design and Development courses that I have taken with him and his continuing support of the Games and Learning Lab, made it possible for me to do this particular thesis. Thanks are also needed for Dr. Jamie Payton for serving on my Thesis Defense Committee and to Dr. Richard Souvenir for encouraging his class to do the study, and taking the time to give us feedback on the game.

Special thanks go to Katelyn Doran and Andrew Hicks who were my undergraduate assistants during the summer of 2008. Without their assistance, the game would not have the vision and strength of purpose that it has now. Michael Eagle, without whom I would still be stuck in a CodeDom, I owe a special thank you to as well. The previous summer, Michael was the lead on figuring out what we could get XNA and C# to do and for that, I am forever grateful. I also owe a debt of gratitude to everyone else who works in the Games and Learning Lab for being there with relevant insights, endless playtesting, and help whenever needed.

Finally, I owe a huge debt of gratitude to the DarkWynter team members – Jason Hardman, Subhir Rao, Priyesh Dixit, and Hunter Hale. Were it not for all the hard work

and dedication in the Games Studio class, we would not be the collective owners of the DarkWynter 3D game engine and I would never have been able to write this thesis. A very special thank you to both Jason and Subhir, both of whom supported and continued development on the engine to get it into a user friendly state, even after the class turn in ended, and both of whom provided me with invaluable advice and support when I decided to torture C# and XNA into doing something Microsoft absolutely never intended for it to be able to do.

This work was partially supported by the National Science Foundation Grants No. CNS-0552631 and CNS-0540523, IIS-0757521, and the UNC Charlotte Diversity in Information Technology Institute.

TABLE OF CONTENTS

LIST OF TABLES	x
LIST OF FIGURES	xi
LIST OF ABBREVIATIONS	xiii
1. INTRODUCTION	2
1.1 Motivation	2
1.2 Prior research	3
1.2.1 Video Games as Course Work	4
1.2.2 Educational Games	8
1.3 Goals and Hypothesis	14
2. PRIOR GAME2LEARN RESEARCH AND PROTOTYPES	16
2.1 Summer Prototypes	18
2.2 Fall Prototypes	22
2.3 Spring Prototype	25
3. <i>ELEMENTAL</i> : THE RECURRENCE – GAME DESIGN	29
3.1 DarkWynter Code Design	30
3.2 Game Design: Recursion	31
3.3 Code Descriptions	33
3.4 Quest Descriptions	35
4. <i>ELEMENTAL</i> : THE RECURRENCE – IMPLEMENTATION	40
4.1 Nuclex Game Control	41
4.2 Compiler Controls	42
4.3 Challenge Objects	44

4.4 Event System	46
4.5 HUD Redevelopment	48
4.6 Custom Game Objects	50
4.7 AI Modifications	50
4.8 Study Conclusions	65
4.9 Instructions Controller	52
5. THE STUDY	53
5.1 Pilot Study Design	53
5.2 Pre and Posttest Analysis	56
5.3 Compiler Answers	59
5.4 Qualitative Survey Answers	61
6. CONCLUSIONS	65
6.1 Study Conclusions	65
6.2 Project Conclusions	66
6.3 Future Works	67
6.3.1 Future Studies	67
6.3.2 Engine and Game Improvements	69
7. REFERENCES	72
8. APPENDIX	76
8.1 Code Challenge 1 – Scaffolding Code	76
8.2 Code Challenge 2 – Scaffolding Code	77
8.3 Code Challenge 3 – Scaffolding Code	78
8.4 Example EventSystem XML	79
8.5 EventSystem Explanation	80

8.6 Pretest Code and Illustration	82
8.7 Pretest Questions	83
8.8 Posttest Code and Illustration	84
8.9 Posttest Questions	85
8.10.1 – IRB Paperwork - Protocol Approval Application (IRB)	86
8.10.2 - Informed Consent	98
8.10.3 – Professor Recruitment Email	101
8.10.4 – Recruitment Poster	102
8.10.5 – <i>EleMental</i> : The Recurrence’s Game Design (Brief)	103

LIST OF TABLES

Table 1: Student Perception of Intimidation by Peers	5
Table 2: Average Scores Per Question Pretest to Posttest	25
Table 3: Qualitative Responses Tournament vs. Squee	26
Table 4: Level 2 DFS Instructional Dialogue	37
Table 5: Level 3 Telephone Call Metaphor Sample	39
Table 6: In Game Instructions	52
Table 7: Languages Previously Used	55
Table 8: Video Games Participants Like to Play	56
Table 9: Pre and posttest results (N= 16)	57
Table 10: Game Log Averages	59
Table 11: Favorite aspects of <i>EleMental</i>	63
Table 12: Suggested game improvements	63

LIST OF FIGURES

FIGURE 1: Dorm Room	16
FIGURE 2: Cyber Café	16
FIGURE 3: The Temple	17
FIGURE 4: Ceiling of the Temple	17
FIGURE 5: Saving Sera's Egg Drop Quest	19
FIGURE 6: Unlock the Door Scroll	20
FIGURE 7: Dialogue from The Tournament	23
FIGURE 8: Squee Mission	24
FIGURE 9: Wu's Castle For Loops	27
FIGURE 10: Ele Icon	31
FIGURE 11: Thought Icon	31
FIGURE 12: Success Message	32
FIGURE 13: Cera's Dialogue Box	33
FIGURE 14: The Coding Interface	34
FIGURE 15: Game Error Message	35
FIGURE 16: Error Output Window	36
FIGURE 17: HUD Mini Map	37
FIGURE 18: Visualization of a DFS stack showing each node	38
FIGURE 19: DarkWynter Engine Component Diagram	41
FIGURE 20: Compiler Control Flow	44
FIGURE 21: Challenge Objects	45
FIGURE 22: Event System Flow	47

FIGURE 23: Heads Up Display Flow	49
FIGURE 24: Pre and Posttest Scores Per Participant	58
FIGURE 25: Time in Game and Attempts vs. Test Scores	61
FIGURE 26: Average Survey Responses on a 5 Point Likert Scale	62
FIGURE 27: Traditional vs. Gaming Assignments	64
Figure 28: Game Entry Point	104
Figure 29: Pretest Survey	105
Figure 30: Coding Interface and Hello World	106
Figure 31: Game Thought	107
Figure 32: Successful Pickup of a Thought	107
Figure 33: Overhead View	108
Figure 34: Game Over Score	111
Figure 35: Posttest	112

LIST OF ABBREVIATIONS

AI	Artificial intelligence
API	Application programming interface
ASCII	American standard code for information interchange
CGC	Current game conditions
CS	Computer science
DARPA	Defense Advanced Research Projects Agency
DFS	Depth first search
DLL	Dynamic link library
FPS	First person shooter
FTP	File transfer protocol
GUI	Graphical user interface
HUD	Heads up display
IDE	Integrated development environment
IRB	Institutional Review Board
MMORPG	Massive multi-player online role playing game
M.U.P.P.E.T.S	Multi-user programming pedagogy for enhancing traditional study
NWN	Neverwinter Nights
NPC	Non player character
POC	Proof of concept
RAPT	Reality and programming together
RPG	Role playing game
SBE	Static board evaluator
UT	Unreal Tournament

XML	Extensible markup language
XNA	XNA stands for Microsoft's toolkit for making games

1. INTRODUCTION

Game2Learn is an innovative project designed to leverage video games to create enjoyable, “playable” programming assignments. Dedicated to the ideal of gaining and retaining computer science (CS) students, the Game2Learn group aims to make the entire CS experience for undergraduate students a more pleasant and productive experience. This thesis presents my work to create and evaluate a game that allows students to write, compile, and run C# code that can interact with game objects during run time. In this research, we outline the design and development of *EleMental: The Recurrence*, a novel game that provides computer science students the opportunity to write code and perform interactive visualization to learn about recursion through depth-first search of a binary tree. We designed the game to facilitate maximum transfer of learning to writing real programs, while also providing for interactive visualization. We conducted a study with computer science majors to measure the impact of the game on learning and on attitudes toward educational games. Our results demonstrate the enthusiasm students have for learning games and provide insight into how such games should be constructed. In future studies, we plan to measure whether learning from the game is retained over the long term.

1.1 Motivation

Even as the employment opportunities for computer scientists rises to meet the current market demands, the number of students enrolling and graduating from college with a degree in computer science (CS) is on a steady decline (Vegso 2007). Most of the attrition in the CS degree happens early on, typically in the CS1 or CS2 course, and is as high as 40% in some cases (Beaubouef & Mason, 2005). Some of the reasons for this attrition are: lack of preparation in the necessary logic and math skills, miscommunications between instructors and

students, poorly designed instructional blocks and topic coverage, poorly designed programming projects and code examples, and under prepared or poor localized language skilled instructors (Beaubouef & Mason, 2005). If we cannot keep students in the computer science programs, how do we expect to produce enough programmers to meet the rising demand?

Many researchers have discussed the possible benefits for using games in education; the problems in computer science education may be uniquely relevant for solution with instructional video games. We hope to leverage the increased motivation games inherently provide to increase student attitudes toward computing and help them learn at the same time (Squire, 2003). Games provide a way to create and share a chunk of educational content while also making students feel that their computing education is more relevant. In other disciplines, games have been used, to teach students a variety of subjects ranging from training a soldier the Arabic language and culture, to teaching high school students the events that led up to the American Revolution (Johnson, et al 2007; Jenkins 2006). In computer science, most instructional research with games involves students building video games, providing an integrative experience that uses a wide range of computing skills along with professional and soft skills (Bayliss, Stout, 2006; Parberry et al, 2005; Becker, 2001). Our Game2Learn project extends this approach, involving computing students in creating educational games that are in turn used for teaching younger computing students (Barnes 2007).

1.2 Prior research

Game2Learn is an innovative project designed to leverage the benefits of video games to create enjoyable, “playable” programming assignments. Our prior results from the Game2Learn project and our research demonstrate the enthusiasm students have for playing

games to learn computer science concepts as opposed to traditional homework assignments, and suggests some new ways that we can improve such games, such as providing immediate feedback, and tying game outcomes to learning outcomes (Barnes, 2007). In the next section, we present the findings from other researchers who have asked their students to create games as course projects and from researchers who have created instructional games, along with a critical review of both.

1.2.1 Video Games as Course Work

Inside the field of computer science, researchers are leveraging the inherent fun of games to motivate their students to doing better in their classrooms. Professor Katrin Becker in the Department of Computer Science at the University of Calgary presents her solution of having the students program games to learn in *Teaching with Games: The Minesweeper and Asteroids Experience*. In her CS1 class, Becker assigned her students to implement the game Minesweeper and assigned her CS2 class to implement Asteroids, both using simple ASCII graphics. Instead of using a standard inheritance programming assignment, students programmed the games and, because of the immediate and visible nature of the objects (the asteroids and spaceship), the students were better able to grasp the idea of object inheritance. Not only did the students enjoy the chance to work on a game, they were inspired to work harder and longer than on a regular programming assignment, going beyond what the actual assignment called for. They also, through word of mouth, spread their enthusiasm to other students in other classes and new students, having heard the glowing reports, come to class already excited about what they get to do this semester (Becker 2001).

Since Becker's report did not present any empirical data on her findings, or details about the assignments, we don't know what aspects made the game assignments appealing, or

how much students learned by making them. In our work we record all aspects of the game assignment and log student behavior in the game environment, giving us opportunities to learn in more detail what is most effective both for learning and for motivation in games.

Games as a “Flavor” of CS1 by Jessica Bayliss and Sean Strout from the Rochester Institute of Technology presents The Reality and Programming Together (RAPT) program, which is a three course programming sequence covering computer science concepts using video games. The CS1 course, the first of the three, was a ten-week summer distance course in which the students were expected to learn the material on their own and program their games in a team setting. After taking the course, the students were given a placement exam and, out of the thirty-seven students, only three failed the placement exams, which put their failure rate at approximately eight percent. Bayliss and Strout also gave the students qualitative questionnaires, to assess the students’ feelings about the course. In particular, they asked the students about their feelings of intimidation by other students and found that, by the end of the course, the number of students reporting intimidation from their peers dropped after taking the course as shown in Table 1. These results suggest that it may be normal to perceive

Table 1: Student Perception of Intimidation by Peers

	Always	Frequently	Sometimes
CS1 (n=369)	7%	15%	23%
RAPT Survey Week 4 (n=33)	0%	3%	30%
RAPT Survey Week 10 (n=21)	0%	5%	19%

intimidation sometimes in a class, but working with games can help reduce the perception of continual intimidation that can occur in introductory courses (Bayliss, Stout, 2006). Finally, Bayliss and Strout presented several ways that games can be easily integrated into computer

science courses; for example, to teach expressions with role-playing games (RPGs), iterations with game loops, and arrays from 2D grid based games (2001).

A strength of this research is that Bayliss and Strout gathered a significant amount of empirical data during the RAPT program. They were able to show an improvement in the programming skills of their students as well as a lessening of the intimidation factor between peers, addressing two main concerns of professors teaching introductory classes. They also presented a clear list of concepts that can be taught via gameplay. However, the paper did not adequately describe the game assignments and the resulting student products so that others could adopt them. *EleMental* is a way for us to share our work in game-based education so that others can try it in their classrooms. It would be interesting to see what kind of games the RAPT students produced, especially because they showed significant learning gains in taking the RAPT course. This would help the research community better understand the impact of games on learning.

Dr. Randolph M. Jones from the Computer Science Department at Colby College presented his work on a capstone game course in *Design and Implementation of Computer Games: A Capstone Course for Undergraduate Computer Science Education* during the winter 1999 semester. In the paper, Dr. Jones discusses his motivations for the capstone course, presenting several arguments to include student motivation, software engineering exposure, and games as object oriented environments. With a somewhat extensive list of the topics covered in the course, Dr. Jones planned the capstone course to cover the software life cycle – from creation of the idea through the final round of testing the software for bugs. In the class, students were required to review commercial video games, conceptualize their own games, build a new game on top of preexisting game code, develop a game from prototype code, and

finally create their own game as a working prototype. Students were allowed to work in teams and there were seven prototypes turned in at the end of the class, to include a 3D racing game, Blackjack, and a role-playing game (RPG). Dr. Jones then presents his ideas on how to make the capstone course better (start the final project much earlier in the semester and maybe have less homework during) and encourages Colby College to continue the course (2000).

Dr. Jones's paper establishes several reasons why a capstone game programming course can motivate students and provides good ideas for what topics should be taught. However, the paper does not include an evaluation of the approach from the student's perspective nor does the paper present findings on how effective the class was at teaching computer science to the students. Further research is necessary to evaluate the student's perspective on the value of the capstone course as well as how effective the class was. In our research, we collect data to measure both computer science learning and attitudes.

Experience with an Industry-Driven Capstone Course on Game Programming by Ian Parberry, et al, discusses a capstone game programming course offered at the University of North Texas. Parberry, et al, motivate the paper with what game companies want students to know upon graduation, as "fresh outs": working on large team projects, nontrivial game creation, knowledge of the industry, and the proven ability to learn without supervision (2005). Parberry argues that a good capstone game programming course should 1) include a team project where all the students are responsible for creating the game, 2) have students program in the same tools and languages that are prevalent in industry (to wit, C++ and DirectX or OpenGL), and 3) require students to make their own game engines instead of using some of the freely available ones. Parberry's capstone course is taught in parallel with an art course so that the programmers have art assets available to them for their games. Students form their own

groups and choose their own games without constraints from the professors (other than making sure the students can finish the ambitious projects) and, at the end of the semester, the games can be entered into a contest judged by industry professionals. They argue for the course's popularity: the course has high enrollments, most of their advanced graduates have taken the game capstone course, and, in a 2003 survey of underclassmen, approximately half the students polled planned to take game programming (Parberry, et al, 2005).

Having the students work on a large game project with the same tools that they would be using in industry is invaluable experience. This experience report provides more evidence that games are good for recruiting students, and gives instructors ideas for how to create a games course. However, capstone courses alone do not solve the problem of retention in computer science; game approaches could be adapted for first and second year courses, and in all game approaches, more emphasis is needed on evaluating what and to what extent computing concepts were learned.

1.2.2 Educational Games

The work discussed so far involves effects of students programming games; in this section, we discuss learning games. In *Entering the Educational Arcade*, Henry Jenkins, et al, discuss *Revolution*, a game created to teach history. Based on Bioware's Neverwinter Nights engine, the MIT - University of Wisconsin Partnership constructed this massive multiplayer online role-playing game (MMORPG) to teach the events leading up to the American Revolution. Students role-play their characters with other students as well as non-player characters (NPCs), learning the social, economic, class, race, and gender issues of the time, not from an epic story perspective, but from a day-to-day, real life perspective. This should help students relate to the people of the time, and realize the similarities and differences in peoples'

lives then and now (Jenkins, et al. 2003).

At the time of this paper, the MIT – University of Wisconsin Partnership was building the game, having completed a good portion of the artwork and some of the storylines needed for gameplay. The planned game was based on sound educational theories of making learning concrete and personally relevant, but we were unable to find any studies of the game that show its effects on students learning history.

In *Experience with Serious Games for Learning Foreign Languages and Culture*, W. Lewis Johnson, et al, from the University of California, Irvine, presents *Tactical Iraqi*, a software course that teaches language learning via an educational video game, that is, perhaps, the best-known military game to date (2007). Built on the *Unreal Tournament 2003* game engine, *Tactical Iraqi* is currently used by the military to train soldiers in speaking and understanding the Arabic languages, as well as teaching the cultural nuances of the area. Soldiers play the game using preset scenarios to interact with various artificial intelligence agents that are based on real life Iraqi personalities. The military is conducting studies to measure the effectiveness of the game, and have already built improvements for the game based on input from native speakers.

There were two major studies reported from 2 different Marine Corps units playing *Tactical Iraqi*; the 2/7 and the 3/7 (Battalion/Regiment). The 2/7 received classroom instruction and 2 hours of *Tactical Iraqi* a day while the 3/7 took 2 hours of *Tactical Iraqi* per week. After the training, the two units did an training exercise that tasked them to interact with Iraqi role players and the 2/7 did better than any other Marine unit who took the Mojave Viper training exercise. The 3/7 unit was actually divided and limited in their interactions with *Tactical Iraqi* and, after running studies, it was shown that the group that used the entire game

had more learning gains than those with limited access (Johnson, et al, 2007). Further research is necessary, perhaps they could add a control group to test the effectiveness of traditional language and culture learning and also use a more quantitative approach to the data gathering.

Logo is a visual programming language created by Bolt, Barenak, and Newman, a research firm in 1967, and is very similar to LISP (but without the parenthesis), where students direct a “turtle” using commands. In *A Comparison of the Effects of LOGO Use and Teacher-Directed Problem-Solving Instruction on the Problem-Solving Skills, Achievement, and Attitudes of Low, Average, and High Achieving Junior High School Learners*, David Dalton compares the effects of using Logo compared to problem solving instruction with a teacher to normal mathematics instruction. In this paper, 97 students in 7th grade mathematics were randomly assigned to either work with Logo (Group A), work with a teacher (Group B), or receive normal mathematic instruction (Group C). The students were then given several standardized tests (the Comprehensive Test of Basic Skills, the Program Criterion Reference Test, and the Test of Non-Routine Problem-Solving Skills, to name a few). The results from the tests indicated that there were no significant learning gains in basic skills achievements from either Group A or B to Group C (the control group). Interestingly, Group B did score significantly higher than the other two groups on problem solving and Group A showed significant motivation (Dalton, 1986).

From this study, Dalton has shown the value of using Logo in a classroom setting but does warn that the effective learning from Logo may not transfer out into a more traditional setting. However, it is interesting to note that the students who learned the most liked Logo better than the teacher instruction and even though this paper only shows that students’ motivation improved by using Logo and not so much their test scores, it is interesting to note

that even the smart students really liked Logo,. However, Logo is a visual programming language that is used in place of, or in addition to, traditional teaching methods and not a video game that teaches students programming. It is unknown how well learning with Logo, or a visual programming language, transfers to learning to program in a standard environment.

In *Programming by choice: urban youth learning programming with Scratch*, by Maloney, J. H., et al, presents their findings on using Scratch, a visual programming language, with urban youths. Scratch was created at the Michigan Institute of Technology (MIT) by the Lifelong Kindergarten Group in collaboration with UCLA and leverages the idea of the Logo system as discussed earlier to teach students how to program in a visual learning environment. During an 18-month period, 536 projects were collected from the Computer Clubhouse patrons (Latino and African American youths, aged 8-18) who were allowed free access to the Scratch system. Out of the 536 Scratch creations, 425 used custom scripts the youths built themselves, with some advanced programming concepts being used (sequential execution, threading, user interaction, conditionals, etc.) in many of the programs as well. From following the students for 18 months, they noticed increased gains in Boolean logic, variables, and random numbers, to name a few. Maloney, et al, also asked thirty random students several qualitative questions and the biggest surprise was their finding that their students did not realize they were programming at all with Scratch. As they had likened Scratch to a notepad, it is rather unsurprising that they would not quite realize what they are doing with the program (Maloney, et al 2008).

It is very interesting to note that the students who used the Scratch program did not recognize that they were programming a computer to run certain commands and logic in the Scratch environment. However, with the lack of formal training of the students, it is not that

surprising – if they have not been told that Scratch allows them to write programs, it does not follow that they would guess that is what they are doing. The students really seemed to enjoy using Scratch, as did the adult supervisors, and everyone seemed to learn something by it. However, like Logo, Scratch it is a visual programming environment, which allows for free play with the system. It is difficult in such an environment to control or measure what learning occurs, and while it may be appropriate for illustrating concepts in the beginning of a college course in computer science, students will need to learn more advanced concepts and techniques not available in Scratch, and with non-visual languages. It is also unknown whether or not the students who use Scratch are more likely to enter computer science programs, and whether it better prepares them for computing courses.

Using Alice 2.0 as a First Language, by Paul Mullins, et al, from the Department of Computer Science at Slippery Rock University, discusses using Alice 2.0 in an introductory computer programming course to teach object oriented programming. Alice 2.0, created at Carnegie Mellon University, allows students to program using pull-down menus in a 3D environment, similar to the MUPPETS system. Instead of creating robots to battle in an arena, however, Alice 2.0 is mostly a story telling game where students create the stories to be played out. Mullins et al did a comparison study between students who used Alice 2.0 to program and students who programmed in a more traditional C++ programming environment. In the study, retention rates seemed to be slightly higher in classes using Alice, even if the instructor changed to using Alice partway into the semester, than the more traditional classes. The paper reports that students who took the course were more likely to be able to feel like they can solve problems than those who took traditional courses (Mullins, et al. 2008).

The final paper in this literature review, *Motivating OOP by Blowing Things Up: An*

Exercise in Cooperating and Competition in an Introductory Java Programming Course by Kevin Bierre, et al. from the Rochester Institute of Technology, is similar to Alice. MUPPETS stands for The Multi-User Programming Pedagogy for Enhancing Traditional Study and is a game that allows students to develop and interact with visible 3D objects in the game world. Using Java, students can, in the game world, directly edit existing code, create new code, compile and run their code and have their code feedback direct in the form of compilation errors or, in the case of the code being correct, have the changes appear in the 3D environment. As a final team project, students code tanks in the TankBrain modification to MUPPETS using Java to attack or defend against other tanks from other student teams. Students are given detailed instructions on how to program their tanks and a sample tank's code is provided to them for comparison. The sample tank code, however, is deliberately designed to not do well in the game, encouraging students to make the tank better. The final for the class consisted of a tournament between all the student team's tanks. During the final, there were several teams that performed well, one team that, due to a partially allowed hack, demolished the competition, and one team that did rather poorly, having spent all their time getting sound to work in the game (Bierre, et al, 2006). After the final, the students took a qualitative survey asking how well they thought they did in the class, their perception of learning from the class, the amount of work required for the class, and if the out of class assignments assist in their understanding of the subject and the results were compared to a traditional class and a class that used Robocode instead of MUPPETS (Nelson, 2009). There were no statistical significance to student's perception of their grade, the amount of work for the class, and if the assignments helped them learn. For the second question, the student's perception of learning from MUPPETS was significantly higher than of the other two (Bierre, et al, 2006).

While the paper presents the MUPPETS system with the TankBrain modification in a very concise and educational manner, the study on the class was based on a post class survey on a Likert scale (qualitative data). The paper does not present any quantitative research from the class. Further research is therefore necessary to determine the effectiveness and the learning gains from having the students create the code for the tanks. It would also be extremely interesting to see the students' log data from the game as that could help tailor new games to better enhance the learning possible from creating the tanks.

Based on results from Logo and Scratch, there is evidence suggesting that using visual programming languages are helpful in learning how to program (or to think in a logical problem solving manner). With Alice and MUPPETS, there is evidence that using game-like environments for learning to program can be helpful for introductory students. However, in all of these and prior games built at UNC Charlotte, students are still using different methods for coding in these environments than they would see in a traditional integrated development environment (IDE) or, as in their early classroom experiences, notepad and a Console.. The extent of knowledge transfer from these tools into traditional programming environments is unknown at this time. With games such as *EleMental*, we hope to incorporate some of the benefits of game programming into early computing courses, while ensuring a realistic development experience.

1.3 Goals and Hypothesis

Our **goal** is to support learning and improve student attitudes regarding computer science education through a well-designed immersive experience that enables **interactive visualization** while supporting **maximal skill transfer**. This requires that the main game mechanic be a direct, concrete, and interactive metaphor for the concept being taught, and that

students realistically construct programs inside the game itself, without leaving the game window and disrupting game play (Barnes 2007). Students who develop such games will also become better computer scientists, benefiting from the idea that “you learn best what you teach,” however; we are not testing that theory in this thesis.

Our **hypothesis** is that *EleMental: The Recurrence* will improve student learning about recursion and depth-first search. We also hypothesize that students who play the game will have a **positive attitude** toward playing such games for homework as opposed to traditional assignments. We believe these results will be based on the success of providing meaningful challenges and an effective interactive metaphor for tree traversal.

2. PRIOR GAME2LEARN RESEARCH AND PROTOTYPES

The Game2Learn group formed in spring 2005 to create a game for learning computer science. At first, we worked with 3D GameStudio, a game development system by Conitec Datasystems, Inc. to create a game to teach students steganography, the practice of concealing and extracting secret messages within an ordinary image or message. In this particular game, players were UNC Charlotte students charged with decoding messages hidden in images created by a kidnapped artist. Figure 1 and Figure 2 show two of the levels that I created for this project, the player's dorm room and the cyber café where the students would play most of the game.



Figure 1: Dorm Room



Figure 2: Cyber Café

Using 3D GameStudio was challenging, since the documentation was primarily in German, and our team was not very familiar with model and level creation. Ultimately, we determined that parsing student code through DLLs was not practical in GameStudio since the game engine redefined several standard elements of C++, such as strings and file manipulation.

After determining that the functionality we needed was impractical using 3D GameStudio, we began building a game prototype in Unreal Tournament (UT) 2004, produced by Epic Games and Digital Extremes. With our engine change, we also redesigned the game into an expanded massively multi-player online role-playing game (MMORPG), now called

StormHaven, to remove it from the confines of the modern era and specific places that needed to be painstakingly modeled as modeling an imaginary place is easier than modeling something in the real world. We decided to make the game an MMORPG for several reasons – 1) we wanted to turn the game into a collaborative learning experience, 2) we wanted to harness several important aspects of MMORPGs such as built-in communications for the users, and 3) the ability to have ‘instances’ of quests so we could build the game and add ‘instances’ when they were completed. Level creation was considerably easier in UT than in 3D GameStudio, and importing models from other tools, such as the Milkshape models shown in Figure 3 and Figure 4, was faster.

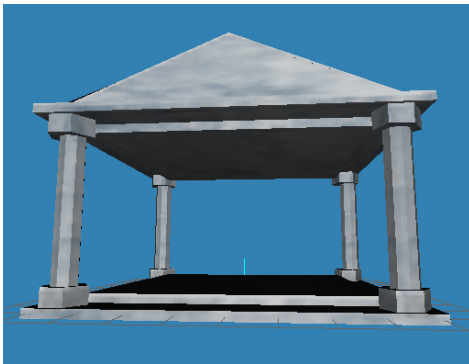


Figure 3: The Temple



Figure 4: Ceiling of the Temple

In this version, I was responsible for the creative design of the game, some of the level design, and for the artwork and 3D modeling of the buildings in the levels as well as some research on integration of a compiler into the game. Although some issues were easier, new problems arose from using a first person shooter (FPS) platform to create a third person role-playing game (RPG), since support was not provided for multiple interaction styles and diverse graphical user interfaces. We again attempted to allow students to write programs during the game, and have them compiled through DLLs. However, this could not be achieved within windows in the game environment using standard programming languages, and would only be

well-integrated using UnrealScript, which is the proprietary strongly typed object oriented programming language (similar to Java and C++) that UT uses. Eventually, we realized that the project's scope was simply too ambitious for a group of students, no matter how dedicated to the idea and the project they were.

2.1 Summer Prototypes

In the summer of 2006, Dr. Barnes narrowed the focus of the Game2Learn project to provide achievable goals within 10 weeks time, as described in (Barnes 2007). I worked with five other summer research students supported through the UNC Charlotte Computing Research Experience for Undergraduates (REU) Site program and the CRA-DMP students. Our two teams of three prototyped the games Saving Sera and StormHaven2: The Catacombs. Saving Sera, which I did not work on, is a 2D RPG built in Enterbrain's *RPGMaker* and was designed to teach students many aspects of computer science, to include for loops and recursion. Playing Arshes, a young hero, students must rescue Princess Sera from an evil wizard. Players earn money and reputation by solving code puzzles in the game, including debugging code for a for loop through interactive dialogue (as shown in Figure 5), rearranging scrambled code, and dragging pieces of a recursive algorithm into correct places in a flow chart.

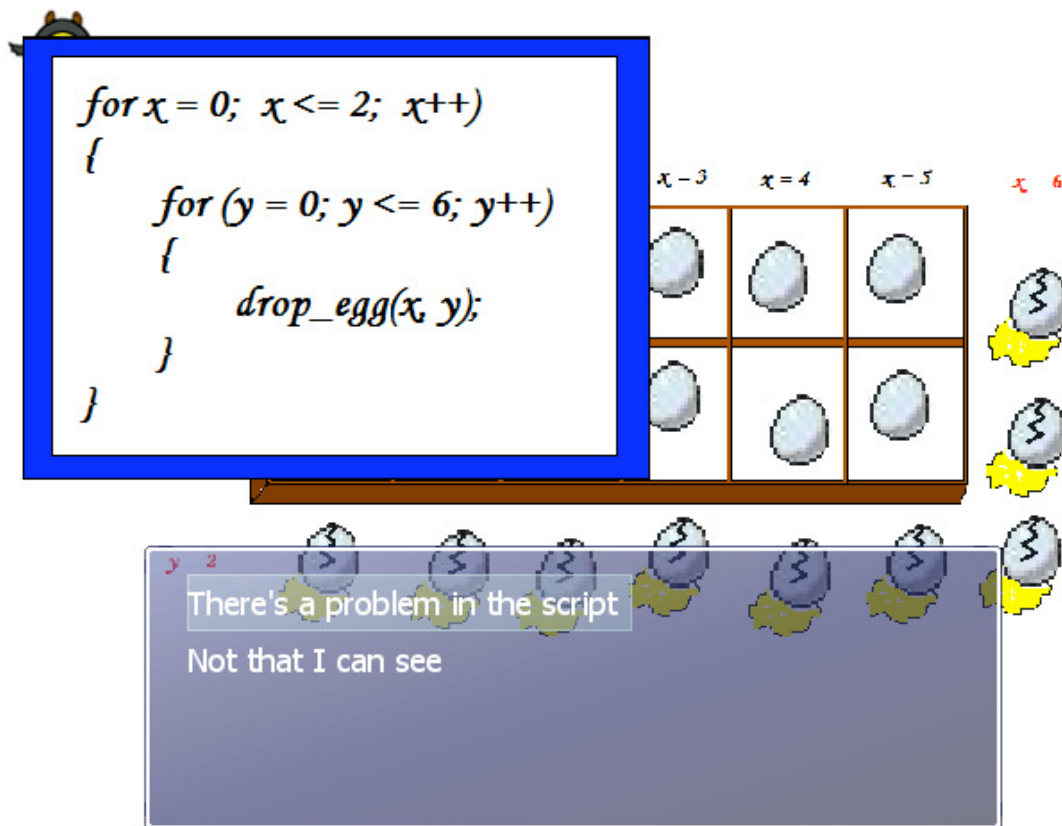


Figure 5: Saving Sera's Egg Drop Quest

The Catacombs, created on the Neverwinter Nights platform using the Aurora toolset, is a 3D RPG where the students create code through dialogue trees with a linear storyline. The object of the game is to rescue two children who have gotten lost in the Catacombs. First the students must break into the Catacombs by coding a nested if statement to open the lock. Second, the students must create code to build a bridge across a chasm, using nested for loops. Finally, the students have to solve a cryptogram using nested for loops to open the door at the end of the Catacombs. Originally, we made two versions of The Catacombs – Grimore and Konijn. The Grimore version uses dialogue from the player's sarcastic spellbook and a dialogue tree to create the code. Konijn uses two interfaces – students had to select the correct 'spell' (code) on a scroll to open the initial door as shown in Figure 6 and then used gemstones

(code snippets) to construct the proper spell (code). After our first run of playtesting and the first study, we decided to drop the Konijn version of the game as students had issues with the interface of using the gemstones to create the code. For this game, I was responsible for creating the first game challenge in each version of the game. This included creating the programming challenge, character interactions, and dialogue. I was also responsible for the main story idea of the game as well as the lead on Neverwinter Nights development, having researched the NWN engine extensively the previous semester.

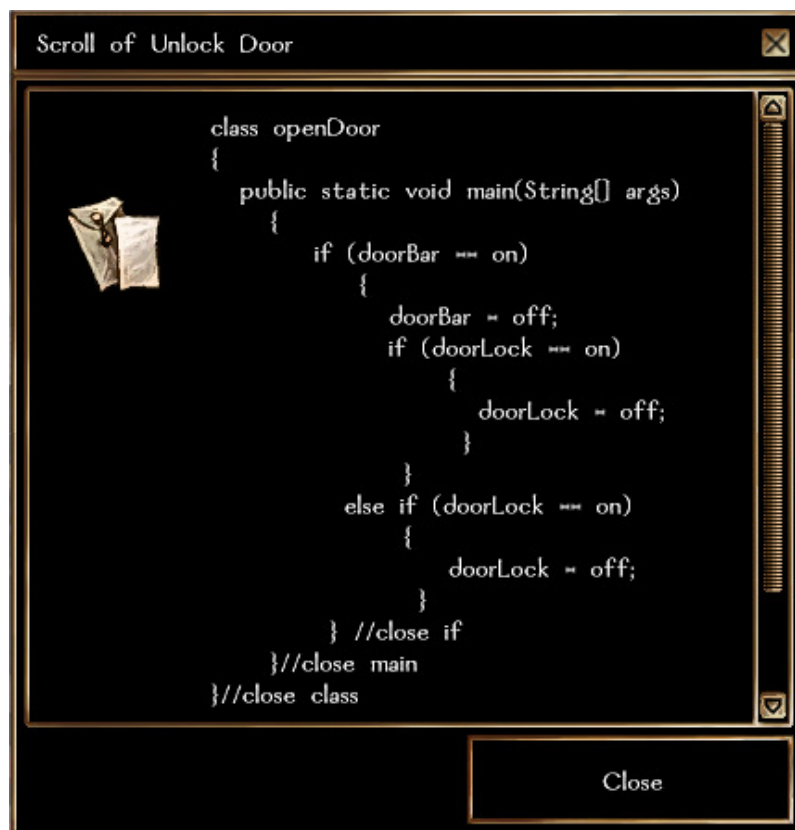


Figure 6: Unlock the Door Scroll

We conducted two studies done on Saving Sera and The Catacombs in Fall 2006. In our first study, we recruited participants who had taken at least one introductory Computer Science course to determine whether they felt the games would be helpful. Participants took a

demographic survey and a pretest, played Saving Sera for twenty minutes, played a version of The Catacombs for twenty minutes, and took a posttest. All interactions were videotaped and we recorded the time stamped logs for each player in Neverwinter Nights. While the participants were enthusiastic about the idea of playing video games for homework, they expressed reservations about the seriousness of these games for homework. In addition, we noticed that most players did not read the game instructions, and we have since learned to incorporate all needed instructions into the game itself whenever possible.

Based on this feedback, we decided that our games would be taken more seriously as homework if the learning tasks were given better feedback and were more integrated with the game score, as explained by Barnes, et al, in *Game2Learn: Improving the engagement and motivation of CSI students*. We built a ranking system into Saving Sera – the player starts at rank seven (the worst) and can decrease to one (the best) by a combination of correct code and time spent on the game. The better the students' rank, the more assistance the NPCs will render in the game (not in the code), allowing the students to purchase better weaponry and supplies. In The Catacombs, we added experience points for correct answers, with more points for fewer incorrect tries. Since experience points control the level of the PC (player character) in NWN, if the students play the game flawlessly, they will 'level up' at the end of the game. If the students provide too many incorrect answers on a single challenge, they turn into a chicken for a brief interval. I also added two cut-scenes, one at the beginning that informed the students of the basic plot outline, and one at the end to show the students' performance. We found that, after our changes, students did learn with the games and take them more seriously (Barnes, 2008).

2.2 Fall Prototypes

During Fall 2007 semester, the Game2Learn team developed two games called StormHaven3: The Tournament and Squee. We tested these games for learning about functions and parameter passing in an introductory computer science course (ITCS1214) taught by Dr. Tiffany Barnes offered at UNC Charlotte. StormHaven3: The Tournament, built on the NWN platform, had a very different concept than any of the other Game2Learn games thus far. Instead of having a concept that students needed to construct code for, we attempted to instruct the students on the concepts and differences of *pass by reference* and *pass by value*. As the creative designer, I came up with the basic plot of the game - the student plays a fighter who must defeat the evil overload in the Tournament. Once again, we split the project into three distinct areas and I was responsible for all of the coding, the dialogue, and the character interactions in the first level, to include cutscene creation. The students first enter the Tournament, using a copy of their signet ring to seal the paperwork (*pass by value*), get to the Tournament (answering multiple questions about parameter passing along the way), and fight the evil overload, either as themselves (*pass by reference*) or by a copy (*pass by value*). Without having specific code in mind, this game is more of an interactive lecture with metaphors explaining the coding concepts, as shown by the dialogue in Figure 7.

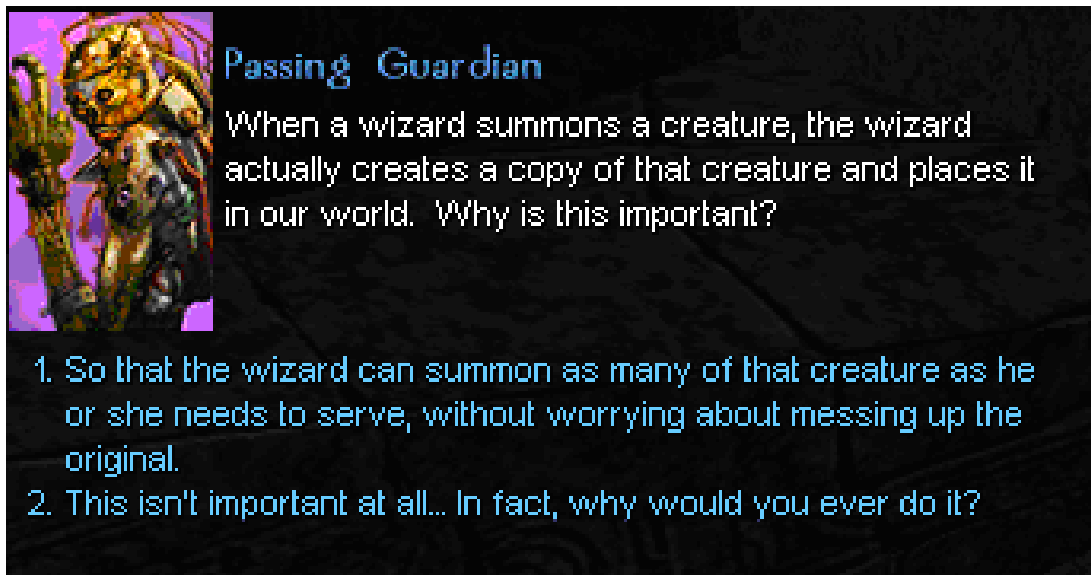


Figure 7: Dialogue from The Tournament

For the first quest, the player may gain up to 50 experience points per correct answer for a total of 100 points, the second is 150 points per correct answer for a total of 450, and the final quest is worth 250 points per correct answer for a total of 500 points. The point system was designed to insure that the player would level up in terms of the Neverwinter Nights rules and allow the player to gain new skills and weapons usage, which would be very important were we to expand this prototype into a larger module. The game starts with a cut-scene to motivate the game and ends with one of two different cutscenes, depending on the outcome of the game.

The same time that we built *The Tournament*, Evie Powell created *Squee*, a spin-off of *Saving Sera*. *Squee* is a RPG developed with Enterbrain's RPGMaker and the player plays a character named Squeebo. In the game, the Squeebos are given a quest to find a certain squeeble (variables) to give to other Squeebos to complete the desired program, thus illustrating the usage, passing of, and scoping of variables. In this game, students had a primary mission, which was to program a "Will Work for Food" sign, as shown in Figure 8. Students had to find the squeeble that would accept a character

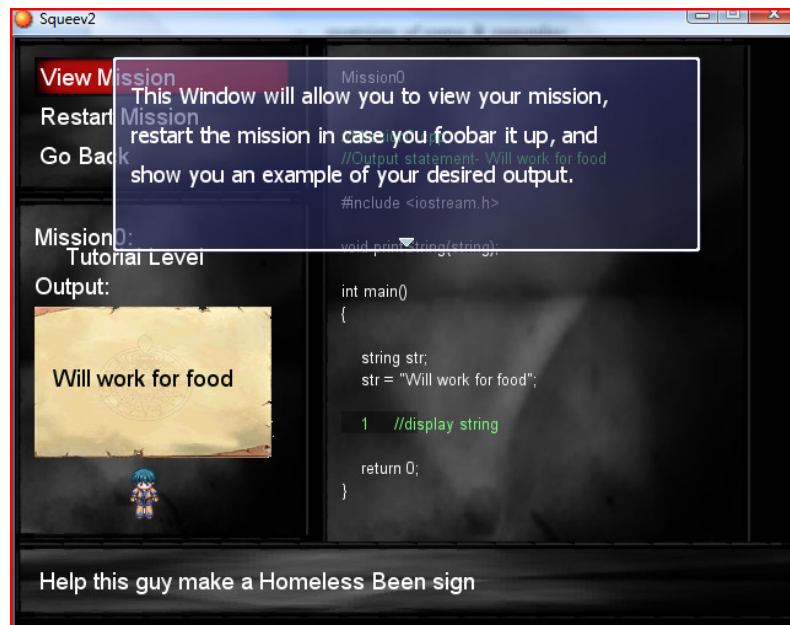


Figure 8: Squee Mission

string as input and display the string on the sign, much the same as a “Hello World” program does, except this time, we pass a variable into the program. There were also several side missions, which included helping other Squeebos through common coding problems and fighting Viruses, which students defeated by answering questions about functions and parameter passing. By the end of the game, the player was expected to be able to: build programs divided into functions, describe the flow of execution in a program with multiple functions, and describe what is meant by the phrase “scope of variable.” Wrong answers given in the game cost the students money, motivating them to think carefully about their answers. This game differs from the other Game2LEarn games because students work directly with code. If the student succeeds at making the correct function call at the correct location in code, the game ends. If they fail, the game indicates that they made an error, but does not indicate the type error type (Barnes 2008).

In the study, participants signed an informed consent form, took a demographic survey

and a pretest of function-related computing concepts, played about 20 minutes of both games took a posttest, and took a survey about their experience. The demographic survey includes information on the participant's race, gender, year in school, major, and gaming habits. The posttest is similar to the pretest, but with the questions rearranged, and the numbers and variable names changed in each question (so that they are isomorphs but not direct copies). Each session took about one hour. We conducted a pre- to posttest comparison for students taking both tests (N=23), and the average scores for each question on these tests is shown in Table 9. Posttest scores are significantly higher ($M = 0.8696$, $SD = 0.1186$) than the pretest scores ($M = 0.6522$, $SD = 0.2372$; $t(22) = 2.472$, $p < .05$).

Table 2: Average Scores Per Question Pretest to Posttest

Question	1	2	3	4	5	6.1	6.2	6.3	7.1	7.2	7.3	8	9	10.1	10.2
Pretest	.57	.14	.32	.57	.71	.25	.25	.25	.54	.61	.54	.35	.57	.14	.32
Posttest	.56	.2	.28	.52	.88	.24	.28	.28	.88	.88	.76	.44	.28	.08	.36

Once the students completed playing the games and the pre and posttest, we also gave them a final survey to determine their enjoyment levels as well as what they thought about the games. Since there were two games in this study, we compared the results of the survey for the separate games as shown in Table 3.

Table 3: Qualitative Responses Tournament vs. Squee

Qualitative Answers	Tournament	Squee
I enjoyed playing	3.04	2.8
This was an easy game to play.	3.52	3.4
Sometimes I was unsure of what I was supposed to do in the game.	3.48	3.36
At first, the game was hard, but it was easy after I got the hang of it.	3.28	2.8
I liked creating code in the game.	3.24	3.08
Overall, this game was helpful in learning computer science.	3.32	3
Overall, the game had a good balance between "play" and "quest" time.	3.6	3.36
I would like to play more quests like these.	2.96	2.8
I was motivated to try hard to answer questions correctly in the game.	3.24	3.04
Sometimes I missed questions on purpose to see what would happen.	2.44	2.24
I guessed at the answers for most of the questions.	2.84	2.64

The survey responses for the two games were often surprising to us. Considering the Tournament is a 3D game built on NWN and that most of our students are casual gamers at best, it was surprising that they liked it a little better than Squee, which is a 2D RPG. Some of our students, however, were very unreceptive to the game and the idea of playing games for homework. About 60% of the students in this survey did say that they would prefer playing games to the more traditional forms of homework while only 44% thought that the games were not serious enough for homework, an improvement over earlier Game2Learn studies. When

allowed to give their own comments, one of the students said, of The Tournament, “it was informative, yet simultaneously slightly entertaining.” In reference to Squee, one of the students said that she liked having, “conversations with the Non-players characters, and the way programming was presented in it.” While this is not an in depth discussion of the study, we are planning on running the study again in the fall and publishing our results in a later paper.

2.3 Spring Prototype

In Fall 2007, Michael Eagle built Wu’s Castle an RPG developed with Enterbrain’s RPGMaker. As presented in *Wu's Castle: Teaching Arrays and Loops in a Game* by Mike Eagle and Tiffany Barnes, Wu’s Castle teaches students the concepts of for loops and arrays in a highly visual and interactive manner (2008). Using a magical fairy and their own programming abilities, students program snowmen to exist, have hats, and arms, etc, by changing the numbers in for loops (Figure 9).

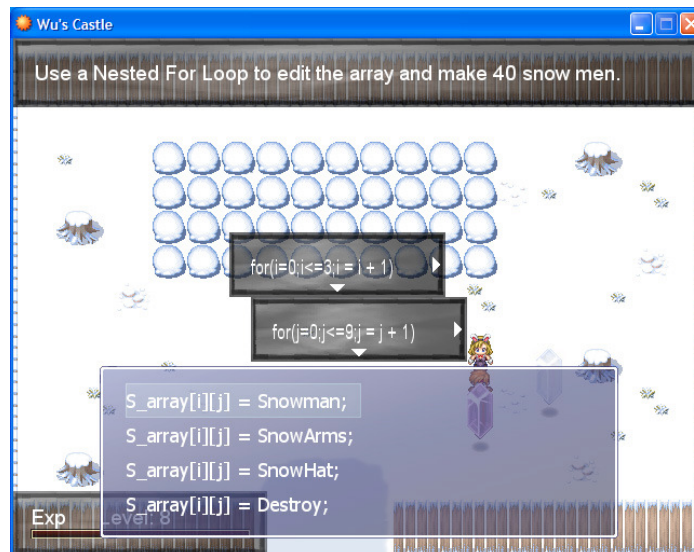


Figure 9: Wu's Castle For Loops

As the for loops modify arrays of snowmen, the students can watch as the for loops control which snowmen get built, or get arms or hats, providing an interactive and concrete visualization of how an array is modified with different loop controls. In the second level, the students walk their player character through an actual nested for loop while the code is step through for them on the side. Once the game was developed, Michael ran the study as extra credit in ITCS 1214. Students were given a demographic survey, a pretest, played the game for up to an hour, a posttest, and a final quantitative survey at the end. Unlike the other Game2Learn games, students could play anywhere they wanted to as the game is an online deliverable, which many students seemed to appreciate. The game has shown positive learning gains in this and later studies with introductory computing students and has been included in our introductory class as a standard laboratory assignment.

Out of all of these Game2Learn games, however, the students never really had to do anything more strenuous than change variables in pre-defined loops, attach the correct variable to the pre-defined program, or be able to read code. None of the games required, or even had the ability, to allow the students to write actual code in the game. None of the games had the ability to run the students' code upon being written, which is one of our stated Game2Learn goals. This thesis, therefore, is dedicated to the idea that having students write actual code (in this case, C# code) which they can compile and run in the game world would be the ideal interface to teach programming in a game environment. The ability to code in the game addresses the concerns that the games are not serious enough for homework, that the students are not writing actual code, and that they are not writing code the same as they would be writing it in the real world.

3. ELEMENTAL: THE RECURRENCE – GAME DESIGN

Designing educational games requires a different focus than general game design; otherwise, we may fall into the trap of designing fun games that have no learning value. As discussed in *Game2Learn: Building CSI Learning Games for Retention* by Barnes, et al, it is best for game designers to first select the target concept for teaching (2007). With the concept firmly in mind, the designers then design the code they want the students to produce. Only after the concept and the target code are designed do the designers begin to develop a game that wraps the concept and the code in a game mechanic that works as a metaphor for the entire game as well as the coding concept. This methodology constrains the game designs and game engine choice to best match the targeted computing concept and metaphor.

After the overall game concept is defined, the designers then tailor the game instructions to support the students in writing code and learning concepts. Game instructions include both how to play and write in-game code as well as educational instruction about the game content. Scaffolding code, i.e. pre-written code provided to help students get started, is designed at this point as well, although we try to limit the overall amount provided to reduce complexity for introductory students. For example, for a complex method that students should use but do not yet need to understand, we provide a correctly formatted method call. In *EleMental*, we have also used scaffolding code to explain the differences in languages, particularly in relating C# to C++ and Java. For more information on how to write scaffolding code for student use, we suggest *Engaging Students in Advanced Computer Science Education Using Game Segments* by Dr. Michael Youngblood, which describes how to make game segments for students. Game segments are complete games with some part of the code removed for the students to fill in, but allow the students to do more than required for an

assignment, especially by making their game look different or have slight differences from others.

3.1 DarkWynter Code Design

We used this Game2Learn methodology to design *EleMental: The Recurrence* to teach recursion. Our previous games focused on teaching introductory concepts; here we wanted to provide a learning game for more advanced computing students with more advanced topics, combined with the capability to write, compile, and run a program within the game environment. There are several reasons that led us to select recursion as the concept for our game: it allowed us to use more complicated data structures (in this case, a binary tree), it enforced the need to create the game in a 3D environment, and the DarkWynter engine allows for real time terrain modification (a player can change the terrain in the game world, adding hills or making valleys, etc in real time), which we could easily adapt for multiple game ideas. In designing the code for the game, we went through several derivations of recursion, to include population growth, Hanoi towers, bridge building, and merge sort, before settling on depth first search (DFS) in a tree structure. With an obvious data structure (a tree), a standard algorithm to code (DFS), and the majority of the implementation needed for the gameplay built into the engine (real time terrain modification), we quickly were able to finalize the code we wanted the students to produce.

The final step was to finalize the scaffolding code that we were providing to the students. Seeing as how we were integrating a compiler into a video game that uses C#, not a language taught in the lower programming courses at UNC Charlotte, we had to design the code the student were going to write to be relatively free of C# quirks. To avoid teaching students more about C# than was truly necessary, we wrote scaffolding code for all of the

coding challenges the students faced (refer to 8.1 – Code Challenge 1, 8.2 – Code Challenge 2 , and 8.3 – Code Challenge 3). After we designed the scaffolding code, we then had to come up with a way to insure that the students would write the specific code that we wanted them to write. This required a parsing method that would check both the students’ code and their output to insure that their code matched what we wanted them to produce. For more details on all implementation, please refer to Section 3. After completing the parsing sections, we then turned our attention to the design of the game.

3.2 Game Design: Recursion

In the game, the students first take a brief pretest to determine their understanding of recursion. After finishing, the students work on a basic “Hello World” coding exercise to get used to the compiler interface and receive basic instruction in C# programming. After they have completed the code challenge, they must traverse a binary tree using depth first search, playing as Ele, the confused college student (Figure 10). At each leaf, there is a Thought for the players to collect to motivate their traversal (Figure 11).



Figure 10: Ele Icon



Figure 11: Thought Icon

Once the student picks up the Thought, the “Go Brain Power” message, as shown in Figure 12 , flashes to give students positive feedback.



Figure 12: Success Message

While traversing the tree, the player is accompanied by Cera Bellum, a mentor who provides instruction along the way for recursion and binary trees through dialogue such as that in Figure 13. In level 2, students must correctly code the traversal for the left side of DFS. After their code is written, Ele (shown in Figure 10), walks through the tree using the student code while Cera explains what the code is doing (the secondary set of recursive instructions). In level 3, students code both the right and left DFS for the binary tree and navigate Ele through the binary tree using the keyboard and mouse. This time, a visualization is provided for the stack calls the recursive algorithm makes. Once the player finishes the game, they take the final survey to complete their journey.

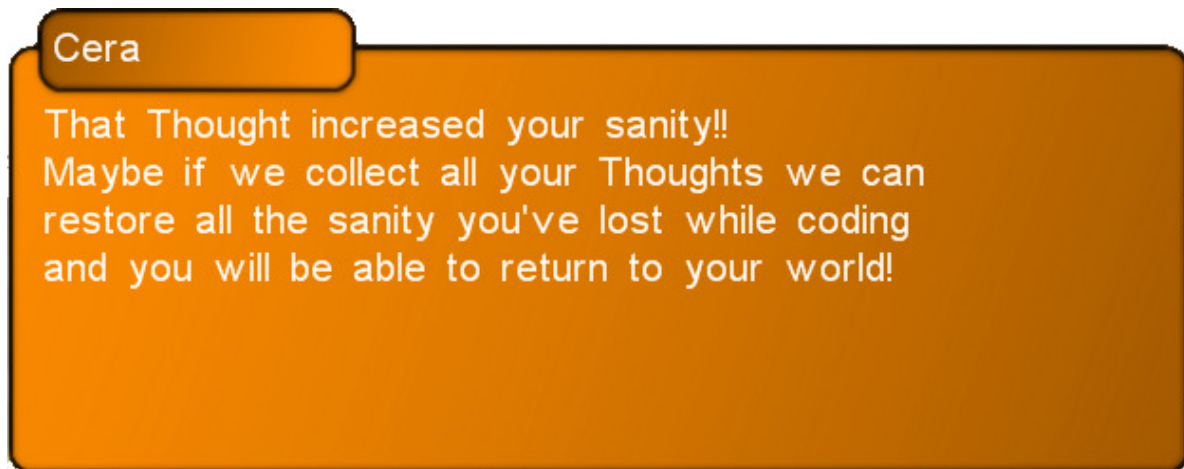


Figure 13: Cera's Dialogue Box

3.3 Code Descriptions

Students write three programs in the game: “Hello World”, finish an incomplete DFS, and write a DFS with more of the code removed. Each program has scaffolding code provided to the students as well as its own particular set of instructions that guide the student into creating the code the way we want them to.

In the “Hello World” puzzle, we explain to the students how the interface works and give them the brief overview of the plot - the students are trapped in their own minds and they have to code their way back to normality. We also explain the only real difference between C# and C++ or Java code that we are going to ask them to write, to wit, `Console.WriteLine` as opposed to C++’s `cout` command or Java’s `System.Out.Println`. Finally, we instruct them to compile and run their code, as well as what to do if their code does not compile. Figure 14 shows the coding interface the students see as well as the properly coded first mission, along with the standard output for the code. The scaffolding code for the Hello World puzzle is shown in 8.1 – Code Challenge 1.

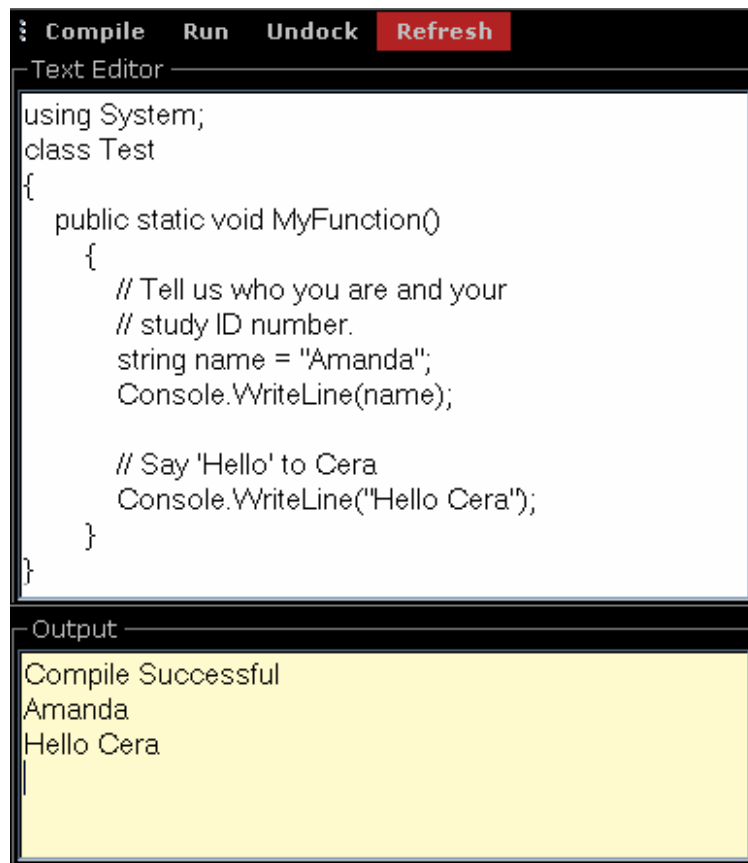


Figure 14: The Coding Interface

In the second puzzle, we provide the students with the scaffolding code to perform a DFS on a tree. Unlike the “Hello World code”, this code is more complex since it performs a DFS on an actual tree using a node system within the game engine. Instead of having the students write the entire program, we simply have them write the traversal code for the left node, which can be modeled after that for the right. 8.2 – Code Challenge 2 shows the code provided.

In the final puzzle, we remove more of the code in the DFS that the students then have to recode. We remove the if statement method call to check if `node.returnLeft()` and `node.returnRight()` return a null value. We also ask the students to construct the code left out of 8.3 – Code Challenge 3 for the actual traversal.

If, at any time during their coding, the students compile incorrect code, they receive the compiler error message, including which line is incorrect. This is the exact message that the students would receive if they were using a standard IDE or console compilation, if their compiler parameters included verbose mode. If the students run code which compiles but has incorrect output or if they did not replace the `node.returnRight()` calls with `node.returnLeft()`, for example, they will receive a customized error message telling them where to look in their code to correct the error. One of the custom error messages a programmer can receive on the third level, for example, is “**Make sure you've set up the if-statement for the first recursive call correctly. You need to check if a right-child exists for the current node.**” Also, anytime a student makes an error, either in the compilation or the execution of the code, the message shown in Figure 15 appears in the game window.

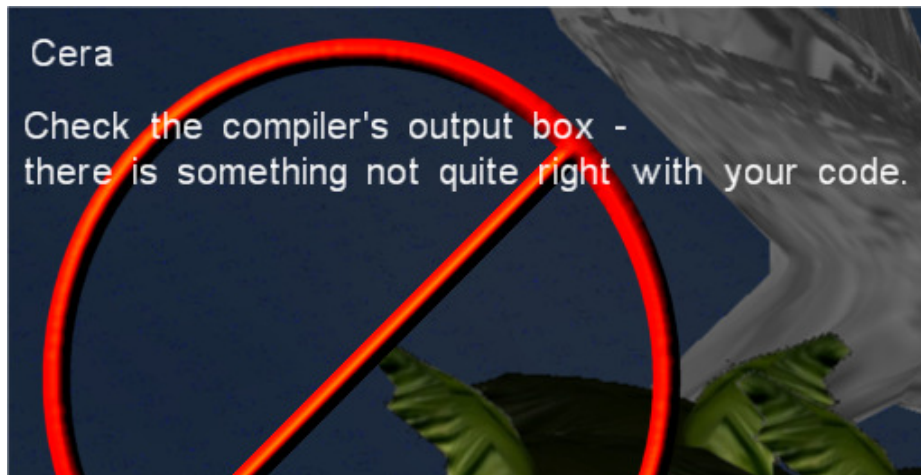
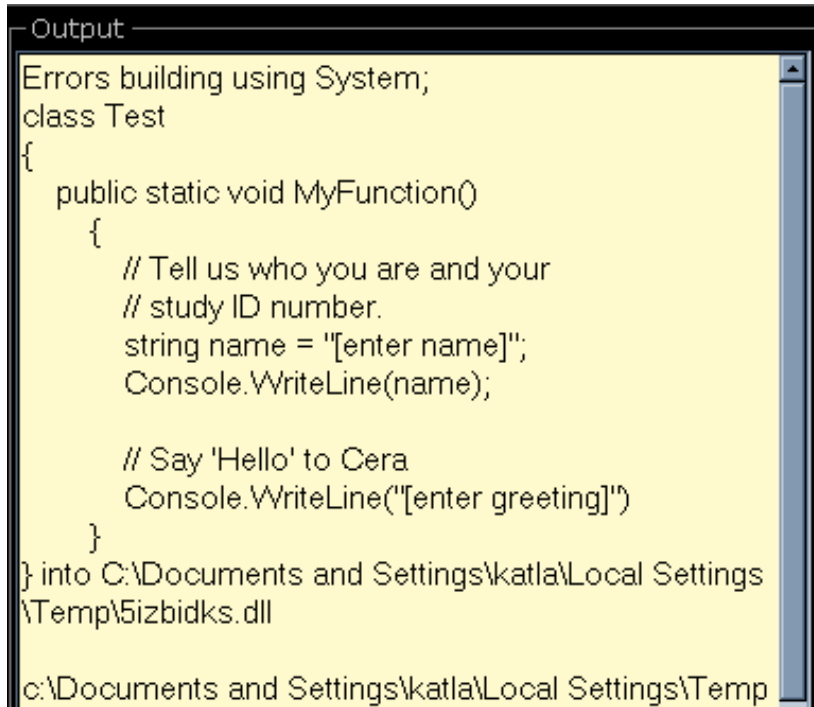


Figure 15: Game Error Message

Figure 16 shows the compiler message that the student receives in the output window of the code interface, note that it shows the verbose error message from the compiler. Using the error messages, the students can figure out where the mistakes in their code are and correct them, the same as they would in any IDE.



```

Output
Errors building using System;
class Test
{
    public static void MyFunction()
    {
        // Tell us who you are and your
        // study ID number.
        string name = "[enter name]";
        Console.WriteLine(name);

        // Say 'Hello' to Cera
        Console.WriteLine("[enter greeting]")
    }
} into C:\Documents and Settings\katla\Local Settings
\Temp\5izbidks.dll
c:\Documents and Settings\katla\Local Settings\Temp

```

Figure 16: Error Output Window

3.4 Quest Descriptions

After the student finishes the code portion in each level, they walk through the tree in DFS order themselves or watch their programmed AI (artificial intelligence) agent do so. In the first level, the students walk through the tree as Cera gives them instructions and hints on how to walk through the tree. For example, Cera says the following “**Do you recognize what we're doing? We are traveling to each of the furthest islands in a set order. Can you think of a name for this type of traversal?**” A moment later, when the student reaches the new node, Cera says, “**Yes! I've got it! We are using a depth-first traversal. For this traversal we travel as far down each branch as possible.**” The letter stored at each node is shown in the first-person view, while a mini-map as shown in Figure 17: HUD Mini Map displays only the tree structure and the student's progress, indicated by the white lines connecting the nodes.

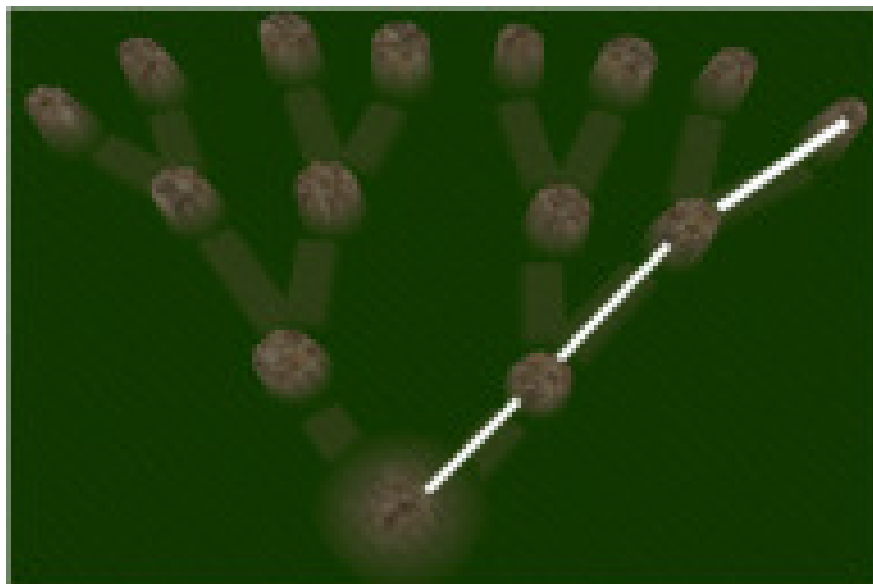


Figure 17: HUD Mini Map

In the second level, after the students write left portion of the DFS, they watch their programmed AI traverse the tree from an overhead view. As the traversal proceeds, dialogue is shown explaining how recursive calls are made to direct the character. 8.2 – Code Challenge 2 – Scaffolding Code shows the code provided and Table 4 has a sample dialogue from Cera for this level.

Table 4: Level 2 DFS Instructional Dialogue

- "As the code runs, `DepthFirstSearch()` is called multiple times. The call is made repeatedly until it returns a value!"
- "Because the previous node returned a value, the Thought can proceed."
- "The Thought has received values for two leaves. Now, it can make its way back by passing these returned values to the previous calls!"

In the final quest, we ask the students to traverse the tree once again, however, this time; we added a stack metaphor and the telephone metaphor to the game. Cera explains to the student how recursive calls to DFS are pushed onto the stack when the student visits that node and are popped off when the student returns from the node. The stack is shown visually as in

Figure 18. As the student visit the nodes in the tree, we add or pop off the stack image appropriate to the scenario, for example, we would add F onto the stack in the first step of Figure 18 and add M in the second step.

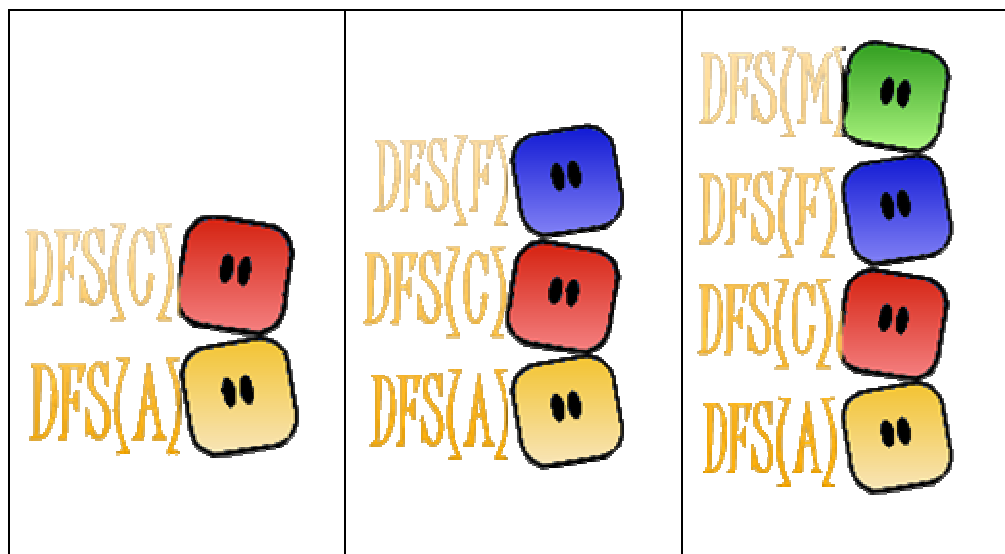


Figure 18: Visualization of a DFS stack showing each node

Table 5 shows sample dialogue from the telephone metaphor we use in the game. We use the idea that the students are calling one another for assistance with their code and each student has to call another student until someone returns with an answer. Then, each student returns answers to the person who first called them. The third level dialogue consists of students calling one another, all the way through the DFS algorithm, starting with A, going down to O, and returning to A once completed.

Table 5: Level 3 Telephone Call Metaphor Sample

- "Think of recursion like this: You need help with your programming homework so you call Bobby (B), but Bobby is confused too."
- "Bobby calls up his friend Eric (E) to see if he can help out, but Eric is kind of dumb. Still, Eric has friends and he calls Jason (J) to ask him for help."
- "Now, Jason returns a response to Eric and the call to Jason pops off."
- "Jason is out of the picture now, but Eric likes to be sure of things so he makes another call."
- "Eric calls Katie. Katie's answer returns to Eric and her call pops."
- "Now Eric has no calls left to make and can return Bobby's call with a response."
- "Now Bobby's call is completed and he can return a value to you. For the purposes of this example your name is Amanda and you're awesome."
- "So now, 'Amanda' your calls have returned a value to you and have popped off. You can continue your calls by calling Cody(C)."

4. *ELEMENTAL*: THE RECURRENCE – IMPLEMENTATION

A little over two years ago, Microsoft released XNA, a hobbyist game development platform that uses C# and the .Net platform. XNA allows game designers to deploy their games on the Xbox 360 by leveraging the managed run time environment designed by Microsoft. XNA Game Studio Express, a free download, harnesses the XNA framework into Visual Studio 2005, allowing game developers the ability to use the Visual Studio integrated development environment (IDE). In the spring of 2007, Dr. Michael Youngblood offered the first formal capstone Game Design and Development Studio course at UNC Charlotte. In this Studio class, the DarkWynter team formed, comprised of four Master's students and one undergraduate, and we created a three-dimensional first person shooter (FPS) game engine. The game engine is capable of real time in-game terrain modification, meaning that a player could raise or lower the terrain at any point in the game. While terrain modification was originally performed with the controller, it was a simple modification to change the inputs from the controller to preset locations, allowing bridges to be built in specified locations at specified times. The main appeal of using the engine, however, was that, unlike using a game engine such as Unreal Tournament or Neverwinter Nights, we were familiar with every aspect of code and could change any aspect of the engine if needed, something lacking in both UT and NWN. We could also add in Windows Forms (.Net framework's graphical application programming interface) as needed, change the heads up display (HUD), and integrate a compiler into the video game, while, at the same time, keeping the text entry by the student separate from the game's update loop. For an overview of the architecture of the engine, please refer to Figure 19. The blue boxes in the figure represent the different project dlls that work together to form a single game and the purple boxes represent the major systems controlled by each top-level dll.

game loop and the game window.

We found a workaround through a DLL for Nuclex game control, an open source XNA modification (2007). Two of my DarkWynter colleagues integrated the Nuclex game control DLL into the DarkWynter engine. The Nuclex game control allows for multiple Form use in XNA, instead of running the single Form in the XNA standard package (2007). Nuclex works by taking the XNA functionality out of the XNA game window and adding that functionality as a thread-safe UserControl. UserControls can be added to a Form or a Panel and, since they are thread-safe, can allow for multiple usage and nesting (Nuclex 2007). The Nuclex game control also overrides both the onPaint() method and the controller methods, allowing inputs into other forms. Therefore, XNA can run in one Windows Form while Windows Forms can be created on the side and on the fly (2007). Access to the Nuclex Control is done via static accessors (as they are thread-safe) and Nuclex replaces the XNA game class.

Since our goal was to allow for in-game coding that included IntelliSense (Windows Autocompletion, documentation, and disambiguation tools for programming) and also separating typing from the game's update loop, my DarkWynter colleagues used Nuclex to create a separate code Form that still had visibility into the engine's code. One of our goals was to include IntelliSense into the code window; however, we were not able to integrate IntelliSense before running the study. We were not able to integrate IntelliSense before running the study but we plan to include it in the future.

4.2 Compiler Controls

The first proof of concept (POC) that we created is the compiler system. Programmers can easily use the .Net CodeDom to build a compiler interface, by compiling C# (or any other .Net language) code and outputting the code compilation at runtime (Microsoft Help, 2008).

The CodeDom, or Code Document Object Model, creates a graph (tree) representation of the code and traverses the tree upon compilation. CodeDom includes a method called the CSharpCodeProvider, which implements the ICodeCompiler. The ICodeCompiler compiles the provided CodeDom tree with the C# standard libraries, allowing us to compile code at runtime (Microsoft Help, 2008). The CodeDom.Compiler includes properties for compilation. In our case, we set the CompilerParameters.GenerateInMemory option to allow the compiler to run the compiled code from memory instead of having to write to and then execute an .exe file, which is the other option (Microsoft Help, 2008). The reason we went with the memory option instead of the .exe was to simplify the process for students. Next, in order to allow students to have access to actual game code, we had to set the CompilerParameters.ReferencedAssemblies.Add() to include DLLs for parts of the engine we wished to be accessible by students. These DLLs include the DWEngine.dll and the DWGame.dll, which gives the students access to anything in either the Engine or the Game. We did not allow access to the DWStream.dll since that is our Renderer DLL, which is very complex, and not needed by the students for anything we would ask them to do. Once the students press the compile button in the GUI, the CompilerControl calls the Compiler class in the Engine (as shown in Figure 20), including any aspects of the Engine or the Game that it needs, to compile the students' code. CompilerControl returns the compiler's output and we use the Console.SetOut method to print it in the output. This allows students to read the exact compiler messages, the same way that they would in a more traditional command line console.

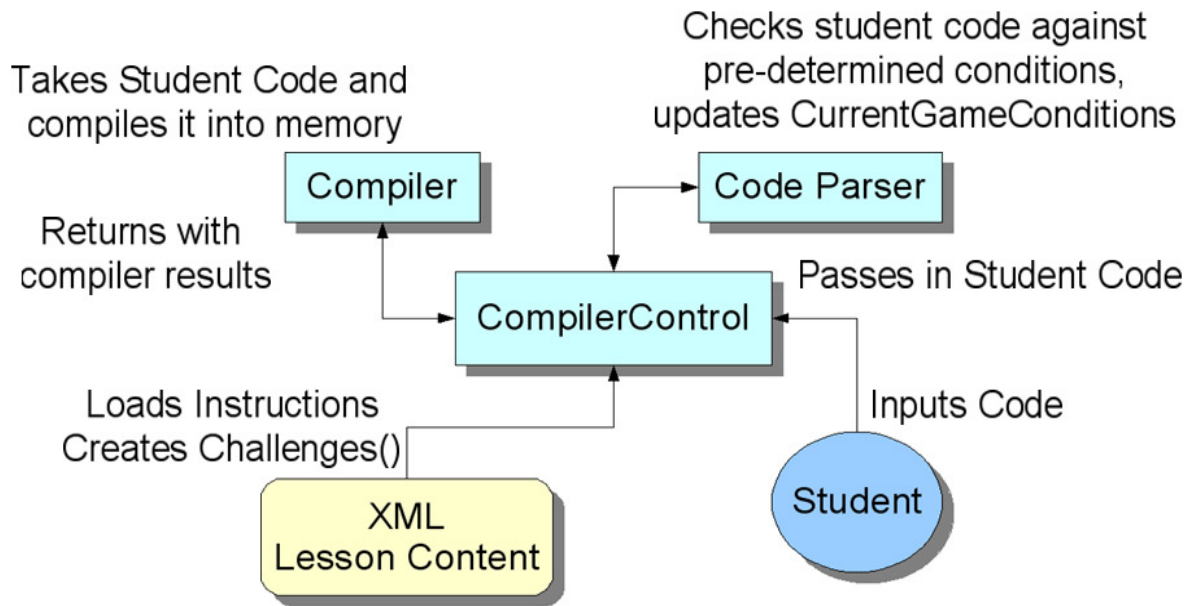


Figure 20: Compiler Control Flow

4.3 Challenge Objects

Challenge objects represent educational activities and contain all the aspects needed to give students a coding challenge. Figure 21 illustrates how challenges are created and how they interact with the game engine. Currently, there are three Challenge objects in the game, created and controlled by the `CompilerControl` class via the `ChallengesAbstract` class. `CompilerControl` loads the Challenges via XML and into a list that the game uses to control which Challenges the student is currently working on. There are three main methods in each Challenge class – `GetScaffolding`, `ValidateStudentCode`, and `RunVisualization`. The `CompilerControl` loads the selected Challenge from the list and the `GetScaffolding` method in each class provides the Compiler Form with the scaffolding code provided for each challenge (refer to the appendices for code for each of the three challenges). Once the students have successfully compiled their code, they have to run the compiled code using the Run button on the GUI.

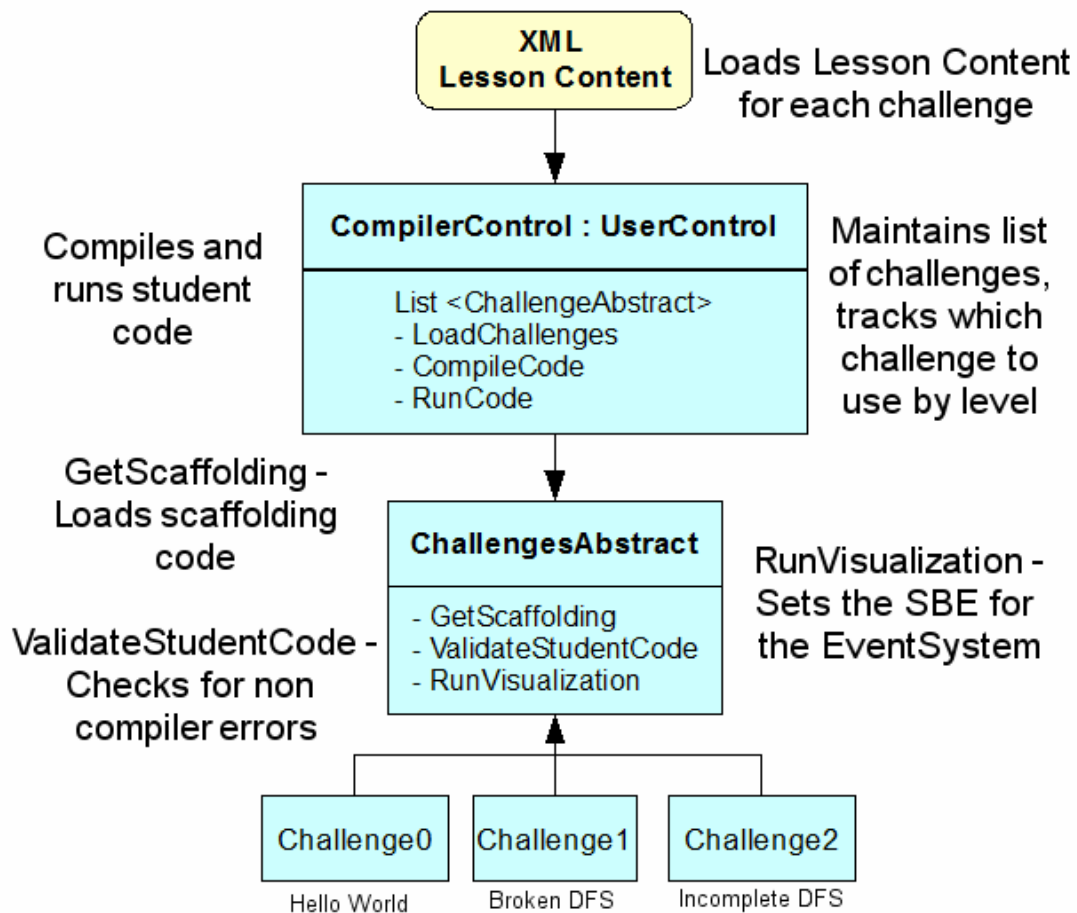


Figure 21: Challenge Objects Flow

In the engine, running the code consists of running the actual compiled code from memory and then passing the information to the `ValidateStudentCode` method in the Challenge classes. The `ValidateStudentCode` method checks the student code to insure that their code matches the code we want them to produce and checks the output of the code to insure that it matches the output of a correctly coded solution. The reason we built in this method was because the Compiler class will compile and execute any code that the students write in the game and, if the students knew enough of the Engine (or encountered the APIs on the Internet); it would be relatively simple for the students to crash the engine, loosing precious data in the process. This method also adds custom error messages to the logging list, letting us know where the students coded

incorrectly, which we pass onto the students as necessary to let them know that they have coded the problem incorrectly for our standards. After passing the validation stage, the Challenge calls the RunVisualization method. This name is an artifact of our first design, where this method was used to build the bridges. Now, RunVisualization sets the flag that the Challenge is complete (allowing the Challenge List to be advanced upon the next level load) and updates the CurrentGameConditions (CGC) CGC is the static board evaluator (SBE), or the condition that the game is in at that precise second, which is used to control the EventSystem.

4.4 Event System

The Event System is the controlling code for much of what occurs in the game. It was initially created by my student mentees in “Wicked Fly Games” (Andrew Hicks, Marietta College, and Katelyn Doran, UNC Charlotte) in the summer of 2008 for their REU work, and I have since modified this system extensively. Originally, the DarkWynter engine was a FPS engine and, but with the addition of the Event System, the engine became vastly more flexible, allowing RPG elements, such as character dialogue and interface (HUD) manipulations. Figure 22 illustrates how the Engine reads in XML for each level; this XML contains all the events for a level.

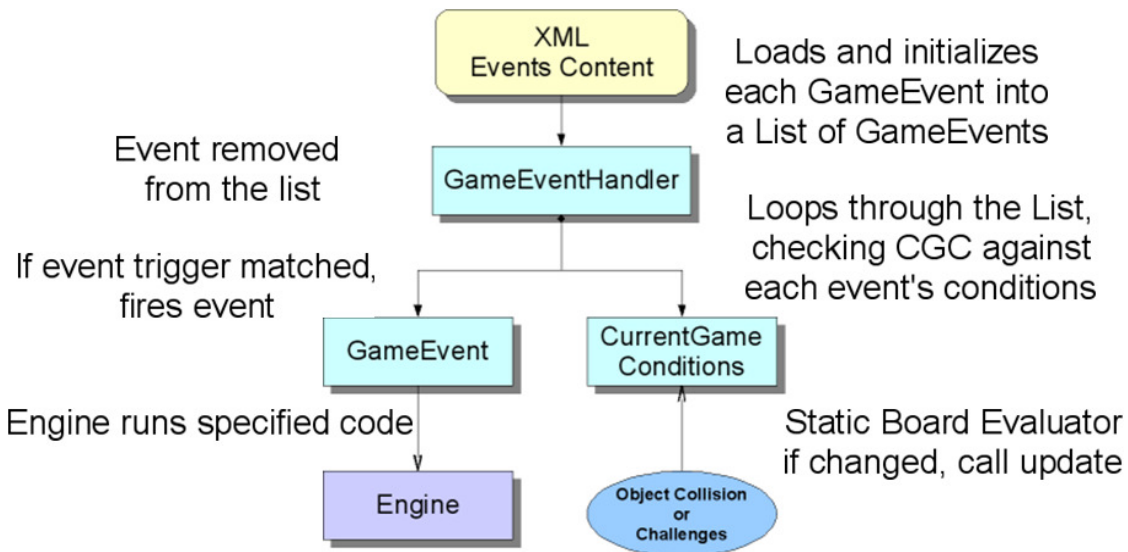


Figure 22: Event System Flow

After loading all of the Events into a controller list and creating the CGC, the Event System waits for changes to be made to the CGC before updating the Event list. Originally, the EventSystem's update was controlled by the Engine's update cycle. However, we have modified the engine so that when changes are made to the CGC, these changes trigger an update to the EventSystem. The CGC can be thought of as a set of triggers. Each Event in the game has a predetermined set of triggers in the CGC; upon update, the EventSystem checks for matches between the CGC and each Event's triggers, and when they match, fires the Event. Currently, there are three triggers being used: the node (the last node visited), coins (the number of Thoughts collected in inventory), and sanity (the player/AI health).

Each Event has a TypeID, individual parameters, and the trigger conditions for the Event. The TypeID allows for creation of the Event object of the specified type and there are, five Event types: AIEvent, TerrainEvent, HUDEvent, DialogueEvent, and LevelChangeEvent. The parameters depend on the type of the Event, for example, an AIEvent needs the destination Vector and the TerrainEvent needs a time limit, start Vector, end Vector, and the radius of the

modification. Once the Event Conditions match the CGC, the Event fires, resulting in either a bridge being built (terrain modification), dialogue appearing on the screen (dialogue event), the AI moving to a preset location (AI Event), the stack modifying on the HUD (HUD Event), the location map changing on the HUD, or the level changing from one level to the next. For a more in-depth explanation of the XML settings for each event type, as well as what each one does specifically; please refer to 8.4 – Example EventSystem XML and 8.5 - EventSystem Explanation.

4.5 HUD Redevelopment

In a video game, the term “heads up display” refers to the part of the user interface that helps the player track his or her progress, inventory, and statistics. For an educational game, the elements required in a HUD are much different than those for an FPS. Therefore, we redeveloped the HUD, particularly to better support the EventSystem and DialogueEvents. Originally, the engine supported up to four players simultaneously and each Human owned its own copy of the HUD. The Human class was responsible for creating, initializing, loading, updating, and drawing its own HUD.

For this project, we decoupled the HUD from the Human class, coupled it with the Renderer, and changed it to a singleton design pattern (public static) to allow access to the HUD from anywhere in the code. Figure 23 shows the interaction between the Engine and the HUD. We added a HUD Update function that calls itself (whenever the HUD is modified) and by Engine (whenever a timed element is needed in the HUD). We created three types of HUD objects (ImageDisplays, ValueDisplays, and TextDisplays, all inherited from ScreenObject), lists to contain them, and modified the HUD Add method to return the position of each HUD

object in its own list. In addition, we improved the look and feel of the HUD.

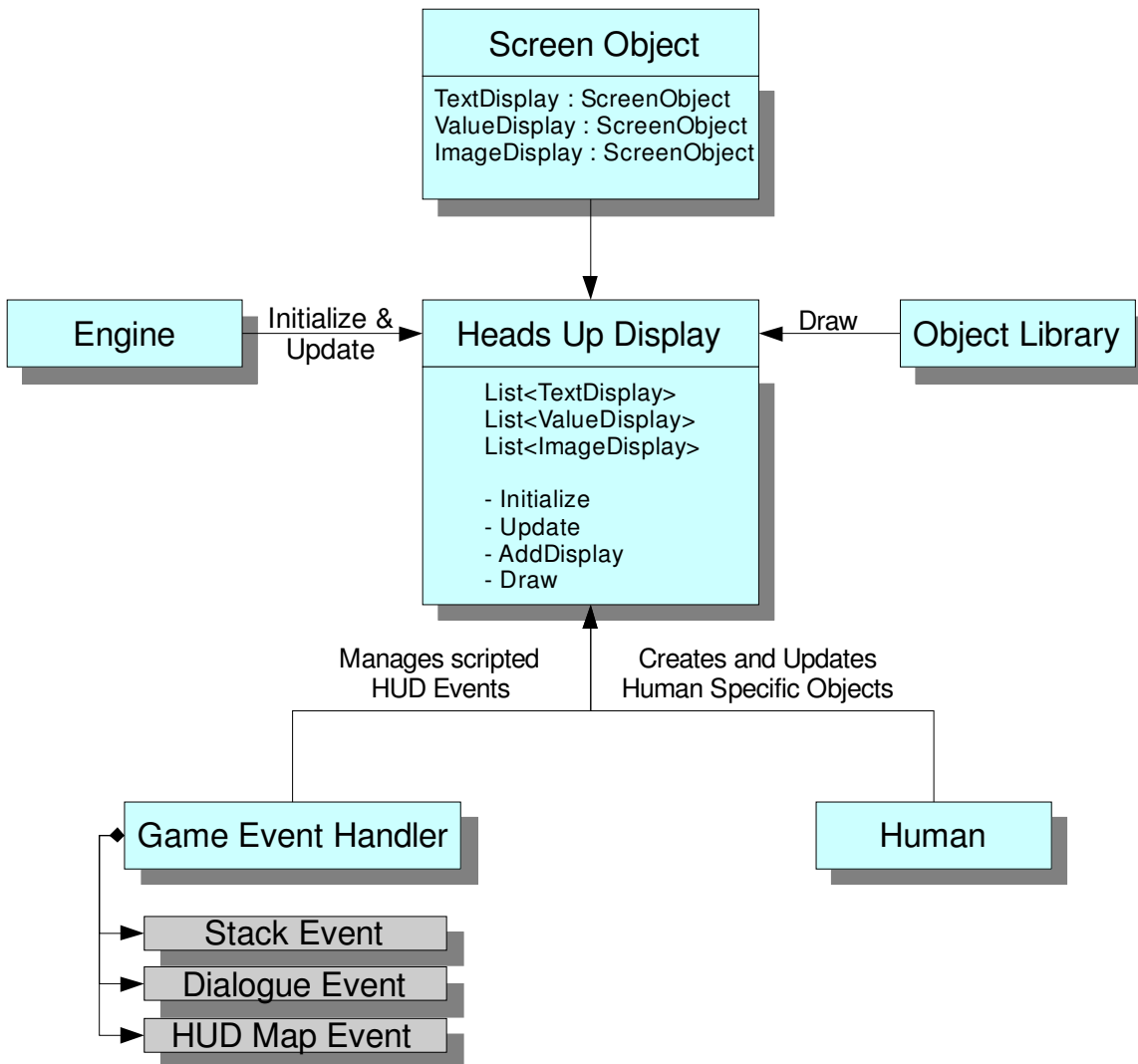


Figure 23: Heads Up Display Flow

Since educational games need instructional and directional dialogue, we added a separate Dialogue Controller that handles `DialogueEvents`. This includes an `Update` method, a `ShowDialogue` method (that shows the dialogue and prescribed background on the screen for a predetermined time limit), and a `RemoveDialogue` method (that removes the dialogue from the screen when the time elapses). Finally, we modified the `DialogueEvent` and the `Dialogue`

Controller load functions to accept a background texture specified in XML to help distinguish different types of dialogue.

4.6 Custom Game Objects

GameObjects were designed to be rapidly extensible as we invented new game objects for the game. Game Objects, as we define them, include any items that the player could interact with, such as the AI, the SkySphere, the WaterPlane, and the Terrain. The GameObjects are stored in the ObjectLibrary, which handles the Initialize, Load, Update, and Draw for each individual GameObject. Three GameObjects - WaterPlane, Arrows, and Thoughts - were developed to facilitate educational game development with DarkWynter. WaterPlane is an object representing the area below the “ground level”, and is used to confine the students to the islands and the bridges between the islands. Collision with the WaterPlane resets the student’s character on the most recently visited island and allows the student to continue the level. To add a level of believability to the “water,” we added a HLSL shader that blends two textures together to give the illusion of flowing water. The Arrows object represents a placeholder in the game, indicating a node players should visit. They are drawn with letters as billboards, or 2-D sprites, where the plane that we indicate in code always faces the player if the player has the billboard in the viewport. Collision with an Arrow sets the last visited node in the CGC to the current arrowID. The Thought object is a collectible item that the students can collect upon collision and are animated to allow for greater visibility.

4.7 AI Modifications

The AI class is a GameObject. The original DarkWynter AI class included a finite state machine that had the ability to find and destroy any Human players on the map. We removed this from the AI, leaving only the walk and idle functionality, and added code to allow the AI to

walk in a straight line (which was not implemented in the original game). We also added the function of rotating in place, which is needed when the AI reaches a leaf node, and adjusting the walk speed for different levels. Currently, we have detected a bug that resets the AI position at some point before loading the XML and AI creation. To work around this, we catch and correct the AI's position in the first pass of the AI's update loop. In future work we plan to find and correct this bug.

4.8 Survey Controller

Having done multiple studies with pre and posttests during the Game2Learn project, we concluded that building a system that collects all of the data in one location is the best way to conduct testing with a game. With that in mind, we added the Survey Controller to the game. When students play the game, they must first take the pretest before continuing into the game environment and take a posttest upon completion of the game. The Survey Controller creates a Form upon Initialize and loads the questions and answer sets, one page at a time, from XML into the Form. Since we load the question and answer sets from XML, we have separated the content creation from the game code, allowing for modularity. In the current pre and posttests, there are five multiple choice questions in each set of tests, all of which can be seen in 8.6 – Pretest Code and Illustration, 8.7 – Pretest Questions, 8.8 – Posttest Code and Illustration, and 8.9 – Posttest Questions. The students can navigate among questions and take the test in any order because we validate that they have answered each question after they press the “Submit” button. Once all questions are complete, the Survey Controller writes the students' answers to a log file. In the future, we will add automatic grading to the survey.

4.8 Instructions Controller

Instructions are delivered through dialogue and the Instructions Controller, a sample of which is shown in Table 6. The Instructions Controller creates a new Form and loads the InstructionSet from the level's XML file, which is designed to be easily modified by instructors. The students can scroll up and down in the instructions, and we can prevent the students from running or compiling their code before they have reached a certain line in the instruction set. In the future, we hope that instructors will use this functionality to create new instructional material and Game2Learn games.

Table 6: In Game Instructions

- Select 'Compiler' from the top of the screen and look at the box that opens to the left.
- That box is a compiler. There is code there to enter your name. Yes, it is written in C# and not C++ but I will walk you through anywhere there is a difference.
- The first thing that I want you to do is tell me your name. Just remove [enter name] and replace it with your name.
- Notice the Console.WriteLine call? It is included twice in this code. This method does the same thing that 'cout' does in C++. It tells the computer what to output to the screen when it runs the program.
- Why don't you use Console.WriteLine command to say hello to me
- Look for the comment 'Say Hello to Cera.' Below this comment is a Console.WriteLine command, say 'Hello' to me where it says [enter greeting].
- Now, you are able to compile your code. If your code compiles successfully, I will know who you are and we can begin working our way out of here.

5. THE STUDY

After receiving IRB approval (protocol number 08-04-22) on October 9, 2008 to run the study, we began recruiting professors and students to take the study, which we officially opened on October 22, 2008. Our recruitment tactics have included emailing professors to announce the study in their classrooms, placing several visible recruitment posters throughout the building, and talking to students. By December 5th, we had 27 participants in the study. Quickly, after we observed that the test scores were decreasing significantly from the pretest to the posttest we realized that the original pretest was too dissimilar from the posttest to determine learning gains. After correcting the pretest, we started the study again on December 9th, ran it through beginning of the following semester. This semester, we have run into the problem of recruiting students as we are asking them to work with recursion, which is typically not started until later in the semester but we have managed to recruit enough students for this study.

5.1 Pilot Study Design

The study design was an exploratory evaluation of the prototype with students that were taking or had already completed Data Structures and Algorithms between October 2008-March 2009 at UNC Charlotte. The purpose of the study was to gather formative feedback and determine the impact and feasibility of using games like *EleMental* for homework. In the study, participants signed an informed consent form, took a demographic survey and a pretest (see Appendix 4) of recursion-related computing concepts, played about 40 minutes of the game, took a posttest (see Appendix 5), and took a survey about their experience. The demographic survey includes information on the participant's race, gender, year in school, major, and gaming habits. The posttest is similar to the pretest, with the numbers and variable

names changed in each question (so they are isomorphs but not identical). Each session took approximately one hour.

The post-surveys were used to gather what testers thought of using games like ours as homework, which quests they preferred, and how balanced the game was in play to quest time. Our primary outcome measurements were the quantitative learning gains and the qualitative responses about whether 1) the games were enjoyable, 2) subjects felt they could learn with them, and 3) subjects would prefer to learn using a game such as these. Gameplay logs were analyzed for the time spent on each game level, as well as the number of correct and incorrect attempts at the code challenges. We planned to use demographic and pretest/posttest results to categorize responses to see if different groups responded differently to the game.

The study was conducted with 43 participants. Twenty-seven of these received an incorrect pretest, so we have data for only 16 participants for a pre- to posttest comparison. Of these, 13 were 18-25 years old and 3 were 25-30. Thirteen of our participants were male and three were female. Two of the students were of Asian descent, 2 were African American, 2 were Hispanic, 9 were Caucasian, and one preferred not to respond. We had 1 freshman, 1 sophomore, 10 juniors, 3 seniors, and 1 post Baccalaureate. Ten were Computer Science majors, 4 were in Software and Information Systems, 1 was in Computer Engineering and 1 was Art Major (Graphic Design) with a minor in Computer Science. The students were also asked what programming languages that they had used previously, as reported in Table 7. As Java and C++ are taught in our first series of programming courses at UNC Charlotte, it comes as no surprise that they are the dominant languages among the students.

Table 7: Languages Previously Used

Answer Options	Response Percent	Response Count
Java	93.8%	15
C++	81.3%	13
HTML	56.3%	9
Javascript	43.8%	7
C	31.3%	5
Visual Basic	25.0%	4
Python	18.8%	3
Basic	18.8%	3
PHP	18.8%	3
Other (please specify)	12.5%	2
Pascal	6.3%	1
Assembler	6.3%	1
SQL	6.3%	1
C#	6.3%	1

We also asked the students what types of games they like to play and their answers are given in Table 8. Interestingly enough for our project, 8 of the students like FPSs, which is what the DarkWynter game engine was originally, and 8 of the students like RPGs, which is what the engine has become with the addition of the EventSystem and some other modifications. Since the students could select more than one answer, and the percent scores are based on the percent of the number of students who selected each game type as a favorite, the percentages do not add to 100%.

Table 8: Video Games Participants Like to Play

Answer Options	Response Percent	Response Count
First person shooter games (e.g. Half-Life, Gears of War)	50.0%	8
Role-playing games (RPGs, e.g. Final Fantasy, Zelda)	50.0%	8
Platform games (e.g. Mario, Sonic the Hedgehog)	43.8%	7
Puzzle games (e.g. Tetris, Bejeweled, Collapse)	37.5%	6
Adventure games (e.g. King's Quest)	37.5%	6
Arcade games (e.g. PacMan, Galaga, Frogger)	31.3%	5
Racing/Driving games	31.3%	5
Fighting games (e.g. Mortal Kombat)	31.3%	5
Massively multiplayer online games (World of Warcraft)	31.3%	5
Rhythm games (e.g. Dance Dance Revolution)	25.0%	4
None, really	18.8%	3
Sports games (e.g. Madden, NBA titles)	18.8%	3
Strategy games (e.g. Age of Empires, Warcraft)	18.8%	3
Simulation games (e.g. Sims, Roller Coaster Tycoon)	18.8%	3

Three of the students never play video games while most play less than 3 hours a week, 1 plays 3-10 hours a week, 2 play 11-20 hours a week and 2 play more than 20 hours a week. Four of the students reported being hardcore gamers, 5 reported being casual gamers, and the rest said they were neither. Because of the small sample size, we did not perform comparisons among the subgroups; however, as we add participants, we will look at the subgroups to determine differences, if there are any.

5.2 Pre and Posttest Analysis

We conducted a pre- to posttest comparison for students taking both tests (N=16), and the average scores for each question on these tests is shown in Table 9. We note that all scores increased except for Q3, and we suspect that this may be a result of students assuming the pre- and posttest questions were exactly the same. We can mitigate the “answer the same multiple choice answer” by randomizing the placement of the answers by student. Posttest scores are

significantly higher ($M = 1.9375$, $SD = 1.12361$) than the pretest scores ($M = 3.1250$, $SD = 1.82117$; $t(15) = -3.048$, $p < .05$). With a p value of approximately 0.008, we can show that there is a significant improvement in the posttest scores after playing the game compared to the pretest scores. The effect size, using Cohen's d , is .78, which is a large effect size, with an α error rate of .05 and a power of .95. On average, the posttest scores increased by approximately .78 standard deviations from the pretest.

Table 9: Pre and posttest results (N= 16)

	Q1	Q2	Q3	Q4	Q5	Total	%
Pretest	0	.5	.63	.38	.44	1.94 of 5	38%
Posttest	.63	.63	.56	.5	.81	3.13 of 5	63%

Figure 24 shows the pretest and posttest result for each participant who took our study. The Y-axis represents the score the student received (5 total possible points) and the X-axis represents the number of the participant. Each participant has two columns in the chart – the light blue column representing their pretest score and the purple representing their posttest score. The participants in the chart are ordered by posttest scores, increasing from left to right.

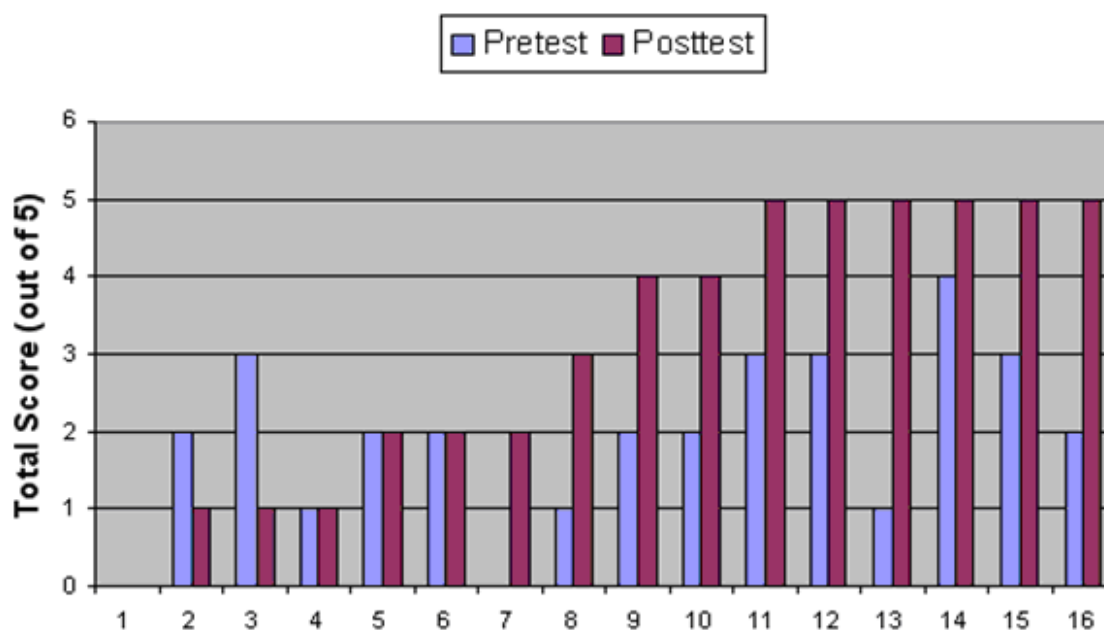


Figure 24: Pre and Posttest Scores Per Participant

As a result of an improper pretest, our study became an abbreviated Solomon's research design that controls for test effects that cause learning from a pretest to the posttest. This 4-group design contains a treatment and control group with both pretests and posttests and has treatment and control groups with posttests only. In our case, we have two treatment groups, Group P with a pretest and posttest and Group NP with just a posttest (N=27). Since our purpose is learning, we have omitted both groups where there is no treatment. We ran an independent samples t-test across the two groups, the ones that took the correct pre and posttests (P) and the no-pretest (NP) group. The means and standard deviations of the two groups were very close to one another, P ($M = 3.23$, $SD = 1.83275$) and NP ($M = 3.25$, $SD = 1.45213$), indicating that there was probably not an interaction for the P group stemming from taking the correct pretest and then taking the posttest.

5.3 Compiler Answers

Table 10 shows the averages and standard deviations over the thirteen correct game logs for each student in group P, which include the overall time spent (minutes: seconds) on the game, and the times and the number of tries students took on each challenge. The logs includes the overall time spent (minutes: seconds) on the game, and the times and the number of tries students took on each challenge. The last student had considerable difficulty because the compiler interprets white space differently than the player expected. We computed the overall time spent on the game as compared to how much of that time was spent in solving the code challenges. On average, students spent 75% of their game time actually working on code challenges. There were no significant correlations between time spent in the game and posttest scores. For 6 of the 13 students with valid logs, their time to solve DFS 2 decreased as compared to that for DFS 1, and for 2 students the time was very similar. While the number of tries to complete DFS2 went up, we believe the decrease in time indicates some learning, since DFS 2 was more challenging. In the future, we plan to do a more detailed analysis of learning times and tries for all study participants.

Table 10: Game Log Averages

Challenge	Average	Standard Dev.
Game Time	32:28	15:02
Hello World	6:57	4:53
HW Tries	1.23	0.44
DFS1 Time	10:02	4:10
DFS1 Tries	2.46	2.26
DFS2 Time	10:32	10:41
DFS2 Tries	4.00	24.68

We also looked at the log files from the game to compare the number of attempts each student had on each programming part and the time that they spent playing the game to their pre and posttest scores. As we only had eleven complete log files for the students for this part, we only compared the students that had the complete log files. We were looking to see if there was a correlation between time spent playing the game and attempts to their test scores. Figure 25 shows the results of the comparison, which, while there was not a significant interaction between time in game and attempts at the programs, does show a trend that students who did well on the posttest did not spend as much time in the game or have as many attempts as their peers. The Y-axis of the chart is the students test scores and average attempt on each problem and the X-axis represents each students' time, test scores, and attempts. Each participant has three columns in the chart – the light blue column representing their pretest score, the purple representing their posttest score, and the yellow representing their average attempts on the programming problem.

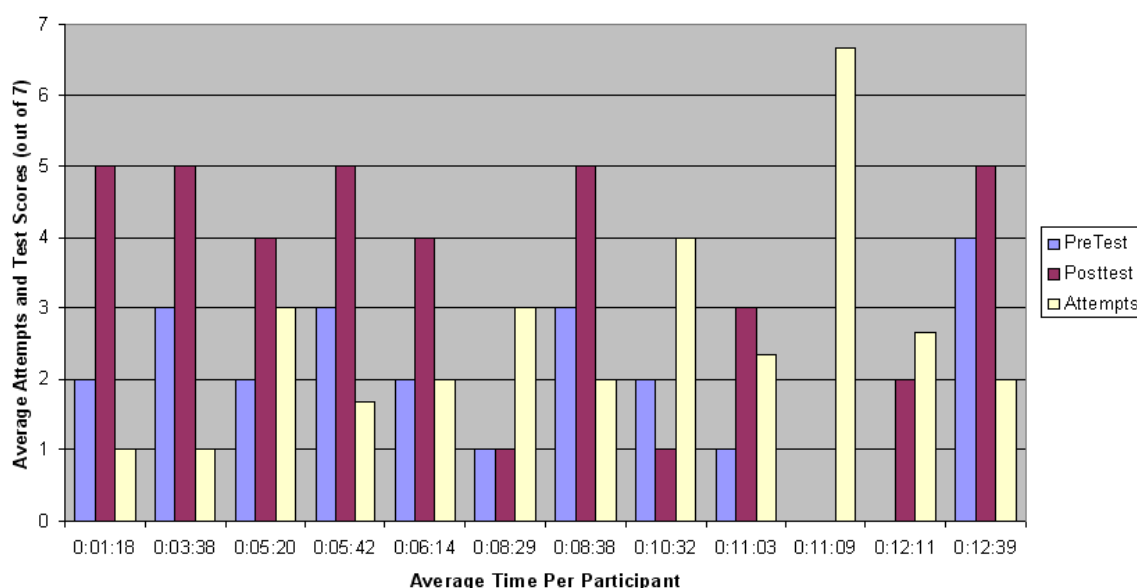


Figure 25: Time in Game and Attempts vs. Test Scores; Participants Ordered by Time Spent

5.4 Qualitative Survey Answers

While we only have valid pre- and posttests for 16 students, 42 students gave us survey responses after playing the game. Out of the 42 students, 34 strongly agreed or agreed that they enjoyed playing the game. As shown in Figure 26, we have averaged the results from the survey questions, where one equals strongly disagree and five equals strongly agree. Thirty-five respondents agreed that they enjoyed creating and compiling their own code inside the game, which is an impressive 88% response. Thirty four of the students agreed that the game was helpful in learning computer science concepts and 32 students thought that the game has a good balance computer science concepts and 32 students thought that the game has a good balance between play and quest time. Thirty-four agreed that the second level with the AI walkthrough was more helpful in learning DFS than the other two levels. Finally, 32 students disagreed that they miscoded on purpose to see what would happen and 29 students disagreed

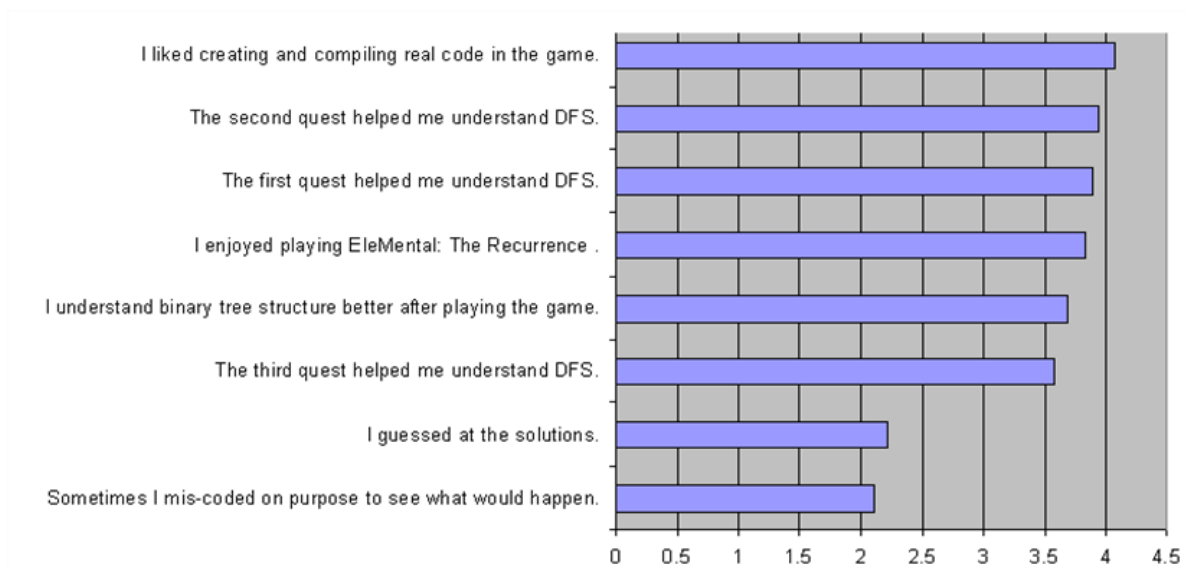


Figure 26: Average Survey Responses on a 5 Point Likert Scale

that they guessed at the solutions. Next, Table 11 shows student responses to the question “What aspects did you like best about the game.” As the 42 students were not limited to a single categorical response, they could list more than one answer in this section, and we categorized them into 67 distinct responses. One student responded, “I liked the whole experience of walking through the paths and then typing the code and then doing it again for repetition. I also liked the example as the tree is traversed,” which we categorized as in game coding, gameplay, and visualization. Another student stated, “I like the fact that I write my own code to make the game work.” While the gameplay was rated the lowest, we believe that is because the students were actually taken with the novelty of in-game coding and the visualization aspects of the game. As for the educational aspects, one of the students said, “It is a great new way to teach computer science concepts to new students.” The hints were also received well; one student said that he really liked the “helpful hints i [sic] was given when i [sic] did not know where to go or what code to write”

Table 11: Favorite aspects of EleMental

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

Forty-two students responded to the item “What improvements would you suggest for EleMental?” and their responses are categorized in Table 12. As the 42 students were not limited to a single categorical response, they could list more than one answer, and we categorized them into 54 distinct responses. Several students suggested we include the player orientation in the mini map for ease of navigation, improve the language of the instructions, and to fix the bugs, such as those that allow students to stray from the path on the binary tree. The three students who suggested that we add audio (though it is already included) did not use headphones. Two students asked for more levels with harder code quests and more game play.

Table 12: Suggested game improvements

Response Categories	By respondents	By categories
Add Audio	7% (3 of 42)	6% (3 of 54)
Improve Mini Map	26% (13 of 42)	24% (13 of 54)
More levels & challenges	10% (5 of 42)	9% (5 of 54)
Improve Instructions	26% (13 of 42)	24% (13 of 54)
Improve Visualizations	9% (4 of 42)	8% (4 of 54)
Improve Engine	33% (16 of 42)	29% (16 of 54)

Finally, we asked the students what they thought about gaming assignments versus traditional coding assignments. Out of the 42 responses, 74%, stated that they would rather play games like this one than code a traditional assignment, while 19% preferred traditional assignments and the remaining students were neutral (as shown in Figure 27). One student wrote, “I believe gaming assignments are better than traditional ones because it’ll make you think more outside the box.” Another student expressed that he believes “Gaming assignments are better because a lot of people enter computer science wanting to do things like gaming, and are disappointed because all they get to do in the first few assignments is write code to calculate tax on different stuff.” Finally, one of our students said, “Gaming assignments...are better suited for my learning style...most traditional assignments have felt kind of useless because they really don’t [sic] do anything besides run...At best a traditional assignment gets you to match your output to the teacher...In this game, at least the code did something visual.”

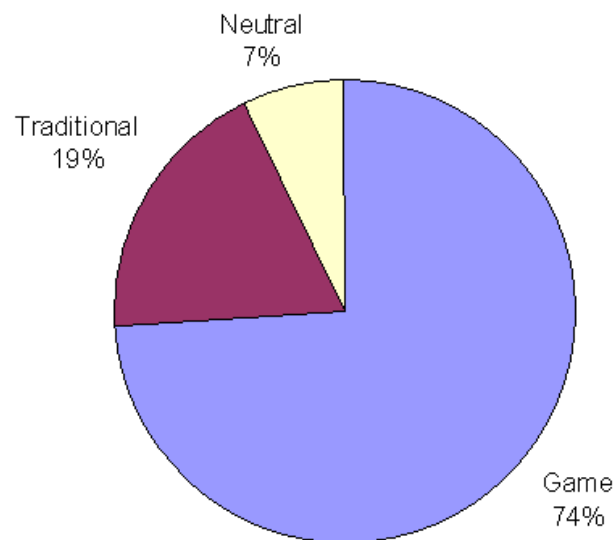


Figure 27: Traditional vs. Gaming Assignments

6. CONCLUSIONS

Creating a game to teach students recursion, a rather notorious subject in the computer science field, is a challenge. We had the standard team communication issues, but since our game programmers included novices, we also had to ensure that everyone learned recursion enough to design and code the game! Comparing this experience to our prior efforts in game creation, we found it much easier to design and create a game for programming with our own game engine on the XNA platform than with a commercial game engine such as Unreal Engine, Neverwinter Nights, or RPG Maker. By using our own student developed game engine, we managed to avoid many of the pitfalls that plagued the Game2Learn project in the early years – namely, not having enough access to the engine to construct a compilation system in the game, not being well versed enough in the engine code to do what we needed to do, and not having the ability to modify parts of the engine (HUD) to show what we needed to show.

6.1 Study Conclusions

Our hypothesis was that *Elemental: The Recurrence* will improve student learning about recursion and depth-first search. We also hypothesized that students who play the game would have a positive attitude toward playing such games for homework as opposed to traditional assignments. In this thesis, we discussed the recursion game design, the game we created and how it was implemented, and presented the results from our study of 42 students playing the game. The study scores from the pretest to the posttest show statistically significant learning gains ($p < .05$) from playing *EleMental* and the qualitative feedback from our participants showed that they were very enthusiastic about the chance to learn with more programming games like *EleMental: The Recurrence*. Although the sample size ($N=16$ for those who took both pre- and posttests; $N=26$ without pretest) was smaller than we would have liked, the

differences in the scores were large enough to show both statistical significance and a large effect size, suggesting that *EleMental: The Recurrence* implements teaching recursion in a game environment in such a manner that students can learn from the game. With the good use of visualization, frequent and obvious feedback both on the HUD and from the compiler, and an in-game compiler that students can use to write their own code to work in our game environment, this game demonstrates many characteristics we feel are important for educational games in computer science. Students played the game for about 40 minutes and most of them were enthusiastic about learning with this game and the possibility of using more such games in learning complex computing topics. We plan to use the results of this study to engage computer science professors in using our student-created, and scientifically validated learning games in their courses.

6.2 Project Conclusions

One of the goals of the Game2Learn project is to enhance computer science education of the student-developers of the games, and this game enriched several areas for myself and my colleagues. In order to ensure that players wrote the expected code, student developers had to learn how the .Net compiler works and how to parse code to ensure that the students were writing the code the way the developers intended. The process improved our knowledge of recursion, compilers, abstraction, polymorphism, and designing complex architectures to handle game interactions. We also learned how to work with DLLs in the .Net environment and that it is definitely better to wait to create said DLLs once that particular part of the code is stable. By using the agile cycle of development, we learned how to rapidly iterate through the software cycle efficiently, using teamwork, good communication skills, self-organizational

skills, and personal and project accountability and management. Perhaps most importantly, student developers learned that good code is no substitute for good design.

6.3 Future Work

Like all Game2Learn projects, the *EleMental* project left us with some very good data on the ability of our game to teach students computing concepts. In the near future, we plan to write a paper describing our design process and giving advice to those who would attempt to make learning games. We would like to expand on the idea, as always, and have several good ideas in the works for future additions to the game. We can use *EleMental*, and in particular, the engine's ability to do real time terrain modifications, to teach students the concept of a heap and provide them with an interesting visual display of the heap being created, modified, and rebuilt, based solely on the student's code input. Another idea is to further expand on the tree that is currently in *EleMental* by allowing students the ability to create and traverse their own trees visualized with real time terrain modification. Finally, we can modify ideas from *EleMental* to teach linked lists and how to traverse, add, and remove list elements to the list in code. Our fully integrated C# compiler enables us great flexibility in both the game types and computing concepts we can teach.

6.3.1 Future Studies

Future studies on *EleMental*: The Recurrence are important to ensuring that our results were accurate and that we did teach the students meaningful concepts about DFS and recursion. We do have some immediate improvements that need to be made, to include making the in-game code harder for the students to complete, but then we need to run the study in an actual ITCS 2214 class to measure student learning in a realistic context. For our first follow up study, we would also like to have half the class play with the compiler in the game and the

other half to play by writing pieces of code that we check in the game, but without using the compiler. Our hypothesis is that students who play *EleMental: The Recurrence* with access to the in-game compiler will show significantly higher learning gains than a comparison group that plays without the compiler. We also hypothesize that students who use the in-game compiler will have better transfer of learning to programming DFS in an integrated development environment (IDE). We believe that the results from that study will help us learn more about what works for teaching computing concepts and how actual programming can reinforce them.

We plan to run a second follow up study to determine the effects of using two or three dimensions in the game. We hypothesize that students who are not familiar with 3D games might perform better in a less complicated 2D environment. However, there may be some interesting positive effects in 3D for students who are more spatially oriented. Therefore, we will include a spatial intelligence test before this study, in addition to our survey on the types and dimensionality of the games students play. We could then compare learning gains and time spent on coding and on navigation, to look for correlations between dimensionality and learning or reduced play time. If we can add position and orientation tracking then we can also measure the amount of turning students need to do to navigate in the game world. We may also be able to simplify the movement controls in the game by using a different controller and programmatically keeping players in the center of the path through the tree. This will help us better understand the interaction of game elements with learning.

We would also like to run a third study to measure the effects of close ties between the core gameplay mechanic and the concepts being learned. For example, in *The Catacombs*, the main gameplay mechanic was walking through a maze and fighting demons, while there were

three code challenges along the way to unlock doors, and code was “magic” in the game. In *EleMental: The Recurrence*, players physically walk and make turn choices based on the algorithm that they are learning. The two main gameplay mechanics here are walking using a DFS algorithm, and writing code to control the physical movement of a Thought using DFS. While both games showed learning gains, it would be interesting to run a study to see if the matched between concepts and gameplay matters for learning gains. Finally, since we are using three distinct metaphors in *EleMental: The Recurrence*, we would like to design and run a study to see if we can detect where the learning occurs in the game, acknowledging that not all students learn the same way. However, if we find that one metaphor is significantly less effective than the others, we could then remove that from the game.

6.3.2 Engine and Game Improvements

Based on our findings in this study, we plan several improvements for *EleMental*, to include most of what the students in the pilot study requested. To better demonstrate the task of depth-first search and the advantages of using recursion, we plan to replace level 1 (“Hello world”) with a small brute-force program to perform tree traversal. We are also thinking about showing the students the ‘backend’ code of the Tree and the Node in the Engine that the students use to build the tree, even if they do not realize it at the time. We might also build a fourth level for the students, where they have to create and use their own Node and Tree classes, which will then be created dynamically for them in the game world. In all levels, we would like to include a visualization of how the actual programmed traversal would work, even when it is incorrect, as we feel that recursion is best taught when students can literally ‘walk through’ the code, the same way that they can in Visual Studios. We may also remove the first-

person perspective for walking through the binary tree, since a top-down view seems to be most beneficial. Finally, we are considering adding “fog of war” (where we hide most of the level detail from the students) to the game to enforce the local vs. global concepts of recursion. What this would do is focus the player on the current path and the choice of left and right paths ahead of them, obscuring the view of lower and higher levels of the tree.

Also based on student feedback, we will add a better indicator on the mini-map to show both position and orientation for the player, and integrate the experience points (XP) better with the game to provide more motivation. We would also like to add an arrow to show players which way to go when they are confused. We designed the stack and telephone metaphor to work together to help students connect local and global behavior for the recursive DFS. However, based on feedback from our data structure and algorithms professor, this area is most difficult for students and could use more elaboration in the game. In our future studies, we also plan to integrate the game as a standard assignment in our algorithms and data structures class, to ensure better study samples and provide all students with the benefit of an alternative form of learning.

Some other engine improvements that we want to make include reducing the time it takes to create a level. Currently, level creation takes a minimum of five hours (for people who know how the engine works) for two developers. This does not include the time it takes to create the dialogue. We would like to decrease this time to two hours for two people, not including the time it takes to develop the dialogue for the level. To do this, we plan to load the Challenges through XML, to make it easier for professors and fellow students to rapidly create new challenges by changing the XML files. We will also look at adding a parser to the game, such as Lex, Yak, or Flex, to assist the developers in parsing the student code.

We are in the process of finishing the level editor, which will allow users to place game objects directly on the game map and then generate the XML. This will allow developers to visually create a level, save the XML, and the game can then load the XML and this will shorten development time considerably. As soon as the Challenges are in XML and we finish the level editor, users will be able to load new levels by creating a grayscale map (currently in use), create the game objects on the world, create new Challenge, Event, Survey, and Instruction XML files within about two hours.

Finally, we would like to attach the engine to a MySQL database for ease in log handling, parsing, and storage. While we are using FTP currently to store the log data, we would prefer to open a MySQL connection to the database and write the log file as a stream instead of writing the log to a text file, which we then FTP, to the server. This will aid us in the future, as it will correct some of the data loss we have already encountered because it will be a complete log up to the point of an eventual engine crash, if one does occur. Also with the database, we plan on a GUI that has prewritten stored queries so that we can look at the particular datasets we want to see without having to write new queries to analyze the game log data. This will allow us to automate a good portion of what we are doing by hand right now and will considerably speed up the information gathering process for later studies.

7. REFERENCES

- 3D GAMESTUDIO. *Game studio Version 8.7*. Conitec Datasystems, Inc. Accessed March 27, 2008. <http://www.3dgamestudio.com/>
- ACM/IEEE-CS Joint Curriculum Task Force. *Computing Curriculum 2001*. Accessed February 13, 2008. http://www.acm.org/education/curric_vols/cc2001.pdf
- AMORY, A. NAICKER, K., VINCENT, J., AND ADAMS, C. 1999. *The use of computer games as an educational tool: Identification of appropriate game types and elements*. British J. of Educational Technology. 30, 4 (1999), 311-321.
- BARNES, T., E. POWELL, A. CHAFFIN, H. LIPFORD. *Game2Learn: Improving the engagement and motivation of CS1 students*. ACM GDCSE 2008.
- BARNES, T., H. RICHTER, A. CHAFFIN, A. GODWIN, E. POWELL. (2007). *Game2Learn: Building CS1 Learning Games for Retention*. ITiCSE2007: 121-125
- BARNES, T., H. RICHTER, A. CHAFFIN, A. GODWIN, E. POWELL, T. RALPH, P. MATTHEWS, AND H. JORDAN. (2006). *Game2Learn: A study of games as tools for learning introductory programming*. Submitted to SIGCSE2007, Kentucky, USA, March 2006.
- BARNES, T., H. RICHTER, A. CHAFFIN, A. GODWIN, E. POWELL. (2006). *The role of feedback in Game2Learn*. Submitted to CHI2007, San Jose, CA, April 2006.
- BAYLISS, JESSICA D. AND SEAN STROUT. *Games as a "flavor" of CS1*, Proceedings of the 37th SIGCSE technical symposium on Computer science education, March 03-05, 2006, Houston, Texas, USA
- BECKER, KATRIN. *Teaching with games: the Minesweeper and Asteroids experience*, Journal of Computing Sciences in Colleges, v.17 n.2, p.23-33, December 2001.
- BEAUBOUEF, T., AND MASON, J, 2005. *Why the high attrition rate for computer science students: Some thoughts and observations*. ACM SIGCSE Inroads Bulletin, 37(2), 2005, 103-106.
- BIERRE, K., VENTURA, P., PHELPS, A. & EGERT, C. (2006). *Motivating OOP by blowing things up: An exercise in cooperation and competition in an introductory java-programming course*. ACM SIGCSE Bulletin, Proceedings of the 37th SIGCSE technical symposium on Computer science education SIGCSE '06, 38(1).
- BIOWARE CORPORATION. *Aurora Neverwinter Toolset*. Accessed March 23, 2008. <http://nwn.bioware.com/builders/>

- DALTON, DAVID W. *A Comparison of the Effects of LOGO Use and Teacher-Directed Problem-Solving Instruction on the Problem-Solving Skills, Achievement, and Attitudes of Low, Average, and High Achieving Junior High School Learners*. Presented at the Annual Convention of the Association for Educational Communications and Technology (Las Vegas, NV, January 16-21, 1986). Accessed April 14, 2009. <http://www.eric.ed.gov/ERICWebPortal/contentdelivery/servlet/ERICServlet?accno=ED267761>
- DARKWYNTER STUDIO. *The DarkWynter 3D Game Engine*. Accessed March 27, 2008. http://darkwynter.com/wiki/index.php?title=Main_Page
- DELWICHE, A. *Massively multiplayer online games (MMOs) in the new media classroom*. Educational Technology & Society, 9 (3), 160-172, 2006.
- EAGLE, M. AND BARNES, T. 2008. *Wu's castle: teaching arrays and loops in a game*. In *Proceedings of the 13th Annual Conference on innovation and Technology in Computer Science Education* (Madrid, Spain, June 30 - July 02, 2008). ITiCSE '08. ACM, New York, NY, 245-249. DOI= <http://doi.acm.org/10.1145/1384271.1384337>
- ENTERBRAIN CORPORATION. *RPGMaker XP*. Accessed March 25, 2008. <http://tkool.jp/products/rpgxp/eng/>
- EPIC GAMES AND DIGITAL EXTREMES. *Unreal Tournament 2004*. Accessed March 27, 2008. <http://www.unrealtournament3.com/>
- GARRIS, R., AHLERS, R., AND DRISKELL, J.. *Games, motivation, and learning: A research and practice model*. Simulation and Gaming, 33 (4), 441-467, 2002.
- GEE, J. P.. *What Video-games have to Teach us about Learning and Literacy*. New York, USA: Palgrave Macmillan, 2003.
- JENKINS, HENRY, ERIC KLOPFER, KURT SQUIRE, AND PHILIP TAN. *"Entering the Education Arcade."* Source Computers in Entertainment, Volume 1, Issue 1. (October 2003): 17-17. ACM Digital Library. ACM Portal. J. Murrey Atkins Library. August 27, 2006. <https://connect2.uncc.edu/>
- JOHNSON, W. L., WANG, N., AND S. WU, *"Experience with serious games for learning foreign languages and cultures,"* in SimTecT Conference., Australia, 2007.
- JONES, RANDOLPH M. *Design and implementation of computer games: a capstone course for undergraduate computer science education*, Proceedings of the thirty-first SIGCSE

technical symposium on Computer science education, p.260-264, March 07-12, 2000, Austin, Texas, United States

MALONEY, J. H., K PEPPLER., Y KAFAL, M. RESNICK, N. RUSK, *Programming by choice: urban youth learning programming with scratch*, in Proc. of the 39th SIGCSE Technical Symposium on Computer Science Education, pp 267--371, Portland, OR, 2008.

MICROSOFT CORPORATION. *XNA Game Studio Express*. Accessed March 27, 2008.
<http://xna.com>

MICROSOFT HELP AND SUPPORT. *How to programmatically compile code using C# compiler*. Revision 3.0. Article ID: 304655. July 30, 2008. Accessed March 12, 2009.
<http://support.microsoft.com/kb/304655>

MOSER , ROBERT. *A fantasy adventure game as a learning environment: why learning to program is so difficult and what can be done about it*, Proceedings of the 2nd Conference on Integrating Technology into Computer Science Education, p.114-116, June 01-05, 1997, Uppsala, Sweden

MULLINS, P, WHITFIELD, D., AND CONLON, M. 2008. *Using Alice 2.0 as a First Language*. To appear in CCSC 24th Annual Eastern Conference 2008 (Hood College, Frederick, Maryland, October 10 and 11, 2008).

NELSON, M. *Robocode*. Version 1.7.1. March 24, 2009. Accessed April 4, 2009.
<http://robocode.sourceforge.net/>

NUCLEX. *XNA Windows.Forms UserControl*. 2007. Accessed March 12, 2009.
<http://www.nuclex.org/news/2007/05/14/xna-windows-forms-usercontrol>

PARBERRY, IAN, MAX B. KAZEMZADEH, AND TIMOTHY RODEN, *The art and science of game programming*, Proceedings of the 37th SIGCSE technical symposium on Computer science education, March 03-05, 2006, Houston, Texas, USA

PARBERRY, IAN, TIMOTHY RODEN, AND MAX B. KAZEMZADEH, *Experience with an industry-driven capstone course on game programming: extended abstract*, Proceedings of the 36th SIGCSE technical symposium on Computer science education, February 23-27, 2005, St. Louis, Missouri, USA

PRENSKY, M. *Digital Game-Based Learning*. New York: McGraw Hill, 2001.

SHAFFER, DAVID, KURT R. SQUIRE, RICHARD HALVERSON, AND JAMES P. GEE. *Video Games and the Future of Learning*. Academic Advanced Distributed Learning Co-Lab. December 10, 2004. August 3, 2006. <http://www.academiccolab.org/resources/gappspaper1.pdf>

- SQUIRE, K. *Video games in education*. International Journal of Intelligent Games & Simulation, 2(1). 2003. Last retrieved March 20, 2009.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.8500&rep=rep1&type=pdf>
- VEGSO, J. Continued Drop in CS Bachelor's Degree Production and Enrollments as the Number of New Majors Stabilizes. *Computing Research News*, Vol. 19, No. 2, March 2007.
- YEE, N. "Motivations for Play in Online Games", *CyberPsychology and Behavior*, 9(6), 2006, pp. 772-775.
- YOUNGBLOOD, G. M. 2006. *Engaging Students in Advanced Computer Science Education Using Game Segments*. *Journal of Game Development*. Vol. 2 Issue 2. 2006 pp. 33--45.

8. APPENDIX

8.1 – Code Challenge 1 – Scaffolding Code

```
using System;
class Test
{
    public static void MyFunction()
    {
        // Tell us who you are
        string name = "[enter name]";
        Console.WriteLine(name);
        // Say 'Hello' to Cera
        Console.WriteLine("[enter greeting]");
    }
}
```

8.2 – Code Challenge 2 – Scaffolding Code

```
using System;
class Test
{
    public void TreeTraversal()
    {
        Tree myTree = new Tree();
        depthFirstSearch(myTree.root);
    }
    public void depthFirstSearch(Node node)
    {
        Thought.moveTo(node);
        if ((node.returnRight() == null) &&
            (node.returnLeft() == null))
        {
            return;
        }
        else
        {
            if (node.returnRight() != null)
            {
                depthFirstSearch(node.returnRight());
                Thought.moveTo(node);
            }
            if ([YOUR_CODE] != null)
            {
                [YOUR_CODE]
            }
        }
        return;
    }
}
```

8.3 – Code Challenge 3 – Scaffolding Code

```

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

    public void depthFirstSearch(Node node)
    {
        Thought.moveTo(node);
        // Check for Base Case
        if ((node.returnRight() == null) && (node.returnLeft() == null))
        {
            return;
        }

        // Recursive calls
        else
        {
            // Travel to node's right child
            if ([YOUR_CODE] != null)
            {
                depthFirstSearch(node.returnRight());
                [YOUR_CODE]
            }

            // Travel to node's left child
            if ([YOUR_CODE] != null)
            {
                [YOUR_CODE]
                Thought.moveTo(node);
            }
        }
        return;
    }
}

```

8.4 – Example EventSystem XML

```

<GameEvent typeID="TerrainEvent" timer="500" startX="19" startY="0" startZ="49" endX
="20" endY="0" endZ="74" modamount="18" node="2" coins="-1" sanity="-1"/>

<GameEvent typeID="TerrainEvent" timer="50" startX="19" startY="0" startZ="49" endX
20" endY="0" endZ="74" modamount="9" node="3" coins="-1" sanity="-1"/>

<GameEvent typeID="DialogueEvent" speaker="Cera"
message="Awesome! You did it!"
texturePath="Content/_textures/compilerSuccess" time="8000"
node="0" coins="-1" sanity="10"/>

<GameEvent typeID="MapEvent" texture="Content/_textures/MapHUD/map2" texX="212"
texY="10" draw="Yes" node="2" coins="-1" sanity="-1"/>

<GameEvent typeID="StackEvent" texture="Content/_textures/StackHUD/StackH.0" texX
="212" texY="10" draw="Yes" node="4" coins="-1" sanity="-1"/>

<GameEvent typeID="AIEvent" ID="5" drawable="false" destX="1500" destY="13" destZ
="5125" node="5" coins="2" sanity="-1"/>

<GameEvent typeID="LevelChangeEvent" node="15" coins="8" sanity="-1"
nextLevel="1"/>

```

8.5 - EventSystem Explanation

Each EventSystem XML tag starts with the element GameEvent, which we use as the class name (type discovery). Red tags denote attributes of the elements.

Trigger Condition Tags (same across all events)

node – The node that the player is currently on.

coins – The number of Thoughts in the player/AI's inventory

sanity – The player's health.

Other Common Tags

typeID – The name of the Event class we are invoking

timer/time – The Time (in milliseconds) for the event to run.

TerrainEvent Tags

startX – The X coordinate where we want the terrain mod to start. *

startY – The Y coordinate where we want the terrain mod to start. *

startZ – The Z coordinate where we want the terrain mod to start. *

endX – The X coordinate where we want the terrain mod to end. *

endY – The Y coordinate where we want the terrain mod to end. *

endZ – The Z coordinate where we want the terrain mod to end. *

modamount – The amount that we want to raise/lower the terrain. *

* All coordinates, unless otherwise specified, are an Vector3 between 0 and size of map, which is capped at 256 pixels.

** Uses the same coordinate system.

DialogueEvent Tags

speaker – The speaker of the message, which will take any string input.

message – The contents of the message, string input.

texturePath – The address of the background image for the text box.

MapEvent/StackEvent Tags

texture – The address of the mini map/stack image.

texX – The X coordinate of the image on the HUD. *

texY – The Y coordinate of the image on the HUD. *

draw – Tells the game to draw or not to draw the image.

* Texture coordinates are a Vector2 capped between (0,0) – the top left-hand corner of the monitor and the max size of the viewport – the bottom right-hand corner of the monitor.

AIEvent Tags

ID – The ID of the AIEvent – used for tracking which AI event we are on.

Drawable – In game edit mode, when this bit is true, then a object is drawn at the location specified by the Vector3(destX, destY, destZ).

destX – The X Coordinate of the destination for the AI or the drawable object. * *

destY – The Y Coordinate of the destination for the AI or the drawable object. * *

destZ – The Z Coordinate of the destination for the AI or the drawable object. * *

* * The destination coordinates are a Vector3 capped between (100, 0, 100) to (8,999, 5,999, 8,999)

LevelChangeEvent Tags

nextLevel – The number of the level that is to run next.

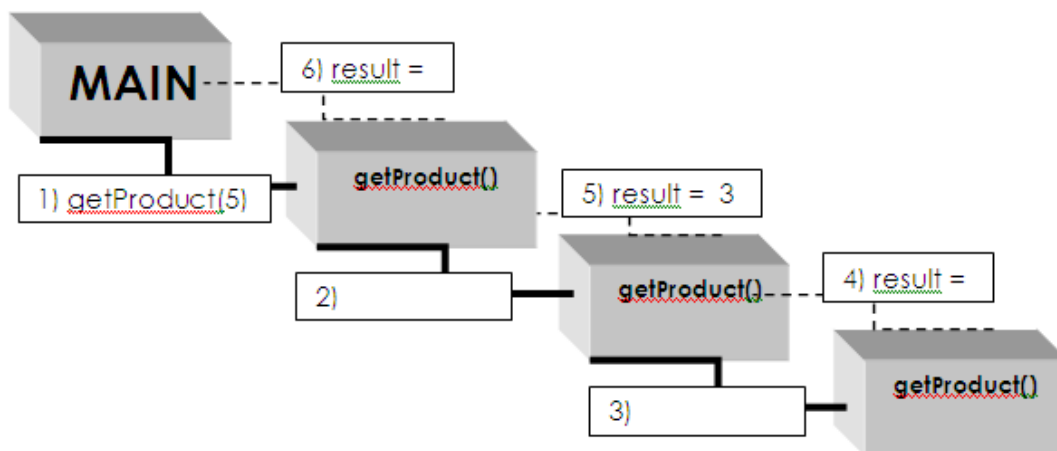
8.6 – Pretest Code and Illustration

```
public int getProduct(int num)
{
    int result;

    if (num == 1)
    {
        result = 1;
    }

    else
    {
        result = num * getProduct(num - 2);
    }

    return result;
}
```



8.7 – Pretest Questions

- 1) What is the value passed to getProduct() in step 2?
 - a. 1
 - b. 2
 - c. 3
 - d. 4
 - e. 5
- 2) What is the value passed to getProduct () in step 3?
 - a. 1
 - b. 2
 - c. 3
 - d. 4
 - e. 5
- 3) What is the value of “result” at step 4?
 - a. 18
 - b. 1
 - c. 0
 - d. 15
 - e. None of the above
- 4) What is the value of “result” at step 6?
 - a. 18
 - b. 1
 - c. 0
 - d. 15
 - e. None of the above
- 5) How would you recursively represent the value of “result” at step 5?
 - a. $\text{result} = 2$
 - b. $\text{result} = 3 * 2$
 - c. $\text{result} = 2 * \text{getProduct}(1)$
 - d. $\text{result} = \text{num} * \text{getProduct}(\text{num} - 1); \text{num} = 2$
 - e. You cannot represent an integer recursively

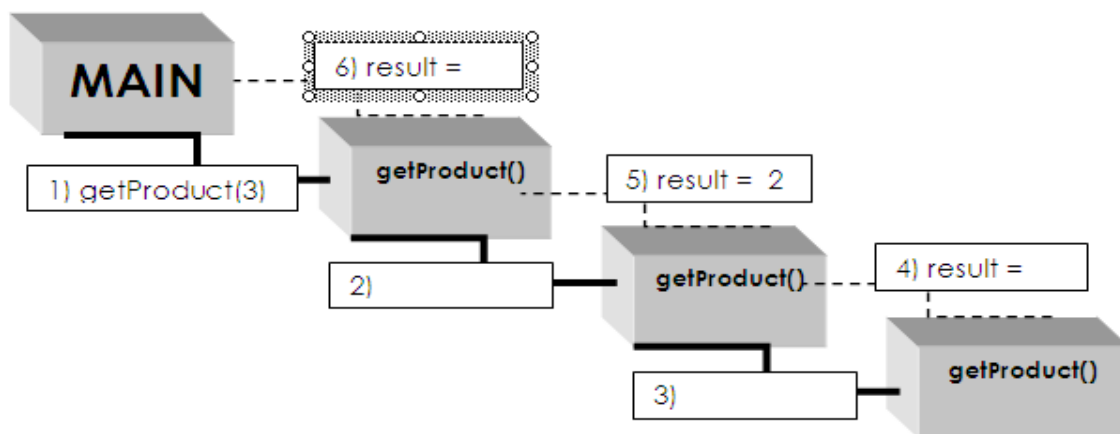
8.8 – Posttest Code and Illustration

```
public int getProduct(int num)
{
    int result;

    if (num == 1)
    {
        result = 1;
    }

    else
    {
        result = num * getProduct(num - 1);
    }


    return result;
}
```



8.9 – Posttest Questions

- 1) What is the value passed to getProduct() in step 2?
 - a. 4
 - b. 3
 - c. 2
 - d. 1
 - e. 0
- 2) What is the value passed to getProduct () in step 3?
 - a. 4
 - b. 3
 - c. 2
 - d. 1
 - e. 0
- 3) What is the value of “result” at step 4?
 - a. 9
 - b. 6
 - c. 0
 - d. 1
 - e. None of the above
- 4) What is the value of “result” at step 6?
 - a. 9
 - b. 6
 - c. 0
 - d. 1
 - e. None of the above
- 5) How would you recursively represent the value of “result” at step 5?
 - a. $\text{result} = 2$
 - b. $\text{result} = 3 * 2$
 - c. $\text{result} = 2 * \text{getProduct}(1)$
 - d. $\text{result} = 2 * \text{getProduct}(\text{num} - 1); \text{num} = 2$
 - e. You cannot represent an integer recursively

8.10.1 – IRB Paperwork - Protocol Approval Application (IRB)

 <p style="text-align: center;">UNCCHARLOTTE</p> <p style="text-align: center;">PROTOCOL APPROVAL APPLICATION</p> <p style="text-align: center;">Institutional Review Board (IRB) for Research with Human Subjects</p>				
PROJECT TITLE	Game2Learn: teaching programming from within a game			
INVESTIGATOR INFORMATION	Name:	Amanda Chaffin	Dept.:	Computer Science
	Title:		Status: Select one:	<input checked="" type="checkbox"/> Student <input type="checkbox"/> Faculty/Staff
	Degree(s): <i>(If student: state degree being sought)</i>	Ph.D. Student	Phone:	(704) 458-3746
	Complete Mailing Address:	9201 University City Blvd, Charlotte, NC 28223	Email:	katla@wulfkub.com
RESPONSIBLE FACULTY	Name	Tiffany Barnes	Dept.:	Computer Science
	Title:	Doctor	Phone:	(704) 687-8577
	Degree(s)	PhD	Email:	tbarnes2@uncc.edu
<p>List all co-investigators below, including those from other institutions.</p> <p>Simply tab to the gray blocks and type in your information. The box will expand as you type.</p>				
Name	Degree(s)	Responsibility on Research Project	Department	Contact Information
Katelyn Doran	Undergraduate Student	Will assist in the design and implementation of the study	Computer Science	Ph: (828) 443-8446
				Email: kedoran@uncc.edu

Investigator's Agreement:

I certify that myself as well as all co-investigators have completed the required UNC Charlotte Human Subjects On-Line Training Tutorial located at <http://www.research.uncc.edu/Comp/human.cfm> and that each of the co-investigators has accepted their role in this study. I agree to a continuing exchange of information with the Institutional Review Board (IRB). I agree to obtain approval before making any changes or additions to the project. I will provide progress reports at least annually, or as requested. I agree to report promptly to the IRB all unanticipated problems or serious adverse events involving risk to human subjects. A copy of the informed consent will be given to each subject if applicable and a signed original will be retained in my files.

 Signature of Investigator

Date

Responsible Faculty Member's Agreement: (If the Investigator is a student)

I certify that, as the student's responsible faculty, I have:

- read and endorsed the materials submitted; and
- completed the required UNC Charlotte Human subjects On-Line Training Tutorial.

 Signature of Responsible Faculty

Date

1. Completion of required Human Subjects Training Tutorial

NOTE: Co-investigators from institutions or organizations not affiliated with UNC Charlotte must either complete UNC Charlotte's required on-line IRB tutorial or provide documentation that similar training has been completed elsewhere.

Amanda Chaffin: Completed human study IRB on Sep 19, 2006 09:04 PM

Katelyn Doran: Completed human study IRB on July 10, 2008 02:54 PM

2. Current or Planned Funding Source (Internal or External)

NOTE: Please submit a copy of methodology section of grant application with protocol application (if applicable).

P.I. of Grant or Contract:	
-----------------------------------	--

Name of Funding Source:		
Grant/Contract No. (if available):		
Grant/Contract or Project Title:		
Attached: Grant Methodology Section	<input type="checkbox"/> Yes	<input type="checkbox"/> No If "NO", please provide explanation in text box below. (Text box will expand.)
No Funding	X	

3. Conflict of Interest

Will members of the research team have financial interest in, receive personal compensation from, or hold a position in an industry sponsoring this study or otherwise have a potential conflict of interest regarding the conduct of this study? If so, please provide explanation below.

No

4. Student Investigators

Indicate if research is for any of the following and provide explanation in the text box below, if needed:

☐ Class project ☒ Undergraduate ☐ Master ☒ Doctoral

This study is being performed by a Ph.D. and an undergraduate students as part of their ongoing research into educational games for computer science education.

5. Purpose of Project

Provide a **brief summary (i.e. 300 words or less)** of the purpose of the project in layman's terms including: background information as necessary, research question(s), and explanation of why the study is needed. Provide the full name/title at least once when using acronyms.

We are developing a game to be used in correlation with Computer Science Data Structures and Algorithm (ITCS 2214 and 2215) courses. This game will be used to introduce and teach computer science concepts within the safe environment of a video game. This innovative learning environment uses familiar game play aspects in order to encourage student learning.

We are developing this game to teach mid level computer science concepts in effort to increase student motivation and engagement in learning to program. We are currently developing an initial prototype containing multiple challenges that focus on the concept of recursion.

For this study, we will be evaluating this prototype to inform the next stage of our game design. We will specifically be focusing on the effectiveness of our in-game lesson plans, as well as initial feedback about

the storylines and game concepts. We will also be studying any learning gains or changes in improvement for the students involved.

6. Enrollment Information	
Expected number of participants:	100
Expected gender representation:	Both male and female, but expect more males due to typical makeup of programming courses where we will be recruiting from.
Expected minority representation:	Will recruit any available minorities, but expect few due to typical makeup of programming courses.
Expected age of participants:	adults, aged 18 and up.

7. Vulnerable Populations	Yes (Target Population)	No (Incidental Inclusion)
---------------------------	----------------------------	------------------------------

Children:	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Non-English speaking:	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Decisionally impaired or mentally incompetent :	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Prisoners, parolees and or other convicted offenders:	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Pregnant women: Select "Yes" if study is about pregnancy, pregnant women and/or the fetus or neonate.	<input type="checkbox"/>	<input checked="" type="checkbox"/>
UNC Charlotte Students:	<input checked="" type="checkbox"/>	<input type="checkbox"/>

8. Characteristics of the Study Population
List required characteristics of potential subjects and those that preclude participation.
<ul style="list-style-type: none"> • Inclusion Criteria: Describe the characteristics of the study population(s). What characteristics make someone an ideal candidate to participate in your study? (e.g., age, occupation, M/F, etc.) • Exclusion Criteria: What characteristics would make someone ineligible for participation in

the study?

Inclusion Criteria:	Students who are currently enrolled in ITCS 2214 or ITCS 2215
Exclusion Criteria:	Students who are not enrolled in ITCS 2214 or ITCS 2215

9. Health Information

The Health Information Portability and Accountability Act (HIPAA) Privacy Rule governs disclosure of personally identifiable health information (deemed “protected health information” or PHI) by hospitals, physicians, and other HIPAA-defined Covered Entities. PHI is broadly defined to include data on a person’s physical or mental health, health care, or payment for health care. PHI includes, for example, a list of a person’s current medications or a person’s weight, smoking status or date of surgery.

As part of this research study, will you obtain any protected health information (PHI) from a hospital, health care provider, insurance agency or other HIPAA-defined Covered Entity?

☒ No

☐ Yes

If YES, attach the Application to Use Protected Health Information (PHI) in Research form at:

http://www.research.uncc.edu/Files/PHI_usage.dot

10. Summary Checklist – Are any of the following involved?

The items listed below ARE NOT an all-inclusive list of methods or procedures but are intended to provide ‘triggers’ or reminders for you to provide appropriate information in subsequent questions in the application or to provide supplemental materials necessary for the review process.

	Yes	No
a) Will research include use of existing data, research records, patient records, and/or human biological specimens?	<input type="checkbox"/>	<input checked="" type="checkbox"/>
b) Will data collection include surveys, questionnaires or psychometric testing? (submit copy of survey/questionnaire with protocol application)	<input checked="" type="checkbox"/>	<input type="checkbox"/>
c) Will data collection include interviews or focus groups?	<input type="checkbox"/>	<input checked="" type="checkbox"/>

<i>(provide interview/focus group question with protocol application)</i>			
d) Will research include deception or less than full disclosure?		<input type="checkbox"/>	<input checked="" type="checkbox"/>
e) Will research include accessing Student Educational Records?		<input type="checkbox"/>	<input checked="" type="checkbox"/>
f) Will research include a data sharing agreement? <i>(Provide details in Question 11 below.)</i>		<input type="checkbox"/>	<input checked="" type="checkbox"/>
g) Will research include an equipment sharing agreement or contract? <i>(Provide details in Question 11 below.)</i>		<input type="checkbox"/>	<input checked="" type="checkbox"/>
h) Will data collection include:	*Audio Recording?	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	*Video Recording?	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	*Photography?	<input type="checkbox"/>	<input checked="" type="checkbox"/>
*If you answered "Yes" to any of the options in <i>Question H</i> , this information must be disclosed in the consent document AND/OR a separate release consent form. (Sample documents can be obtained from the ORS website or from the Compliance Office.)			

If UNSURE, please review the Guidelines for Usage of Protected Health Information (PHI) in Research at:
<http://www.research.uncc.edu/Comp/chipaa.cfm>

11. Full description of the study design, methods and procedures including:

- the type of experimental design;
- describe study procedures;
- provide a sequential description (explained in steps, phases etc.) of what will be asked of/done to subjects;
- clarify if subjects will be assigned to various groups/arms of the study (if applicable);
- explain what kinds of data will be collected;
- provide details on the primary outcome measurements; and
- explain any follow-up procedures (if applicable).

If you answered "YES" to any of the items in Q #10, please provide explanation/description in this section.

Attach 2 copies of the questionnaire(s); inventories, or scales that will be completed by participants.

This study is a pretest posttest experimental design. We will recruit students from ITCS 2214 and 2215 course. We will visit the courses at their meeting times as well as pass out flyers to recruit the students for our study. The students will sign up for specific times to take the study in the Woodward Computer Lab 335. Researchers will be on hand to answer questions. On the day of the study, the students will complete the informed consent, take a demographic survey, take a pretest, play the game, take a post test, and take a final survey.

Sequential description of subject participation:

1. Subjects will complete the attached informed consent form and demographic survey (about 5 minutes)
2. Subjects will take a pre-test on basic programming skills (namely recursion and trees), as attached, to get a baseline on their skills (about 5-10 minutes).
3. Subjects will play through the game (about 30 minutes).
4. Subjects will take a post-test, as attached, of similar construction as the pre-test to determine any changes in skills (about 5-10 minutes).
5. Subjects will complete the attached final, qualitative survey (about 5 minutes).

Total participation time will be about one hour.

Data collected will include all the tests and the game will log their performance to include correct and incorrect responses in game as well as time taken for each task.

Our primary outcome measurement will be quantitative pre to posttest learning gains; as well as, the qualitative responses that will provide us with feedback on whether 1) the game is enjoyable, 2) subjects feel they could learn with it, and 3) subjects would prefer to learn using a game like this. Demographic and pre-test/post-test results will be used to categorize responses to see if different groups respond differently to the game.

12. Duration of entire study and duration of an individual subject's participation, including follow-up evaluation if applicable, including:

- Provide information on the number of required visits, tests, surveys to be completed, interventions.
- Provide information on the approximate duration of each intervention (i.e., how much time should the subject expect to spend).

This study will require that the students take one pretest, one posttest, and a qualitative survey. The game should last no longer than 40 minutes; the pretest, posttest, and survey should last 5 to 10 minutes each. If the participant has not completed the game within 40 minutes, we will ask them to stop. The overall time for each subject is expected to be 1 hour.

13. Where will the subjects be studied?

If off UNC Charlotte campus, list locations.

Attach 2 copies of letter(s) of permission to conduct the research project from school(s), organization(s) or any off-campus location.

The study will be conducted in the Woodward Hall computer lab.

14. Confidentiality

Explain how you will protect the confidentiality of the data collected. Describe procedures for protecting against or minimizing any potential risks from breach of confidentiality or invasion of privacy. How will you protect the data with respect to privacy and confidentiality? For example:

- Where will the data be stored?
- What security measures will be applied?
- Who will have access to the data? Provide explanation of why they need access.
- If applicable, specify your plans for de-identifying or anonymizing the material if audio/video recordings or photographs will be used.
- If applicable, describe what measures will be taken to ensure that subject identifiers are not given to the investigator.
- If applicable, describe procedures for sharing data with entities not affiliated with UNC Charlotte.
- Provide a timetable for destroying the data and identify how they will be destroyed or provide explanation for perpetual maintenance.

Please note: The IRB expects researchers to access the minimal amount of data to conduct the study and to comply with applicable HIPAA and Family Educational Rights and Privacy Act (FERPA) requirements.

Subjects will be assigned a non-identifying number so we can relate all of their various materials. Their identity will be kept confidential, only being revealed on the consent form.

All data labeled with the participant number will be placed into a password protected database in the Games and Learning Lab. The subject identity will be kept confidential, only revealed in a separate password protected master list that relates names to numbers, which only study researchers will have access to. This file will be password protected even when on portable storage devices. In one year, the master list will be destroyed.

15. Data security for storage and transmission.

Check all that apply.

For electronic data:

Secure network	<input type="checkbox"/>
Password access	<input checked="" type="checkbox"/>
Encryption	<input type="checkbox"/>
Other (describe in question 12a. above)	<input type="checkbox"/>

Portable storage (e.g., laptop computer, flash drive)	
Describe in question 12a. above how data will be protected for any portable device	<input checked="" type="checkbox"/>

For hardcopy data (including human biological specimens, CDs, tapes, etc.):

Data de-identified by research team	<input checked="" type="checkbox"/>
Locked suite or office	<input checked="" type="checkbox"/>
Locked cabinet	<input checked="" type="checkbox"/>
Data coded by research team with a master list secured and kept separately	<input checked="" type="checkbox"/>
Other (describe in question 12a. above)	<input type="checkbox"/>

16. Full description of risks and measures to minimize risks:

Give full descriptions and measures risk factors.

For example:

- psychosocial harm (e.g. emotional distress, embarrassment, breach of confidentiality, etc.)
- economic harm (e.g. loss of insurability), and
- legal jeopardy (e.g. disclosure of illegal activity) as well as
- known side effects of study medication,
- risk of pain and physical injury.

There are few risks associated with this study beyond normal computer usage.

The game is set in a three-dimensional environment which adds the slight risk that some users could experience disorientation.

17. Benefits to subjects and/or society:

The possibility of benefits to society should be clearly distinguished from the possibility of benefit to the individual subject, if any.

If there is no direct benefit to the individual subject, say so. Do not list monetary payment as a benefit.

The Game2Learn games have shown significant learning gains in the previous Game2Learn studies. The students could potentially increase their knowledge of the covered computer science topics.

The research will potentially benefit future students in computer courses by providing a more engaging or interesting method for learning. This in turn may attract or retain more women and minorities in the computer science and information technology fields.

18. Inducements for participation:

If monetary, specify the amount and how this will be prorated if the subject withdraws (or is withdrawn) from the study prior to completion.

None provided by the study personnel.

19. Costs to be borne by subjects:

If there are no costs to subjects, indicate this.

There will be no costs to subjects except for the time involved.

20. Data analysis:

State how the data will be evaluated, indicate where and by whom data analysis will be performed.

The data analysis will be both quantitative and qualitative in nature. We will analyze the game log files and the pre and post test results. We will look for learning gains, and patterns and differences in user behavior. We will also look at the demographic and qualitative results to evaluate the games. All analysis will occur in the Games+Learning lab in the Computer Science Department.

21. Methods of recruiting:

Tell how prospective subjects are contacted. Provide recruitment script (letters, email, flyers and advertising, telephone script, verbal, website, etc.).

The study will be offered to a computer science class (ITCS 2214 and 2215). Flyers will be posted in Woodward Hall, passed out in the classes, and emailed to the instructors of the classes to forward to their students.

22. How will informed consent be obtained?

Give full descriptions and measures for all of the following applicable risk factors:

- Describe the process.
- It is typical to obtain assent from children ages 7-17.
- When the consent of a legally authorized representative is substituted for consent of the adult subject, explain why this is necessary.
- If non-English-speaking subjects will be enrolled, a consent form should be prepared in their foreign language.
- Someone who is fluent in the subjects' language must be available to interpret.

Attach 2 copies of the informed consent document(s) printed on your department's letterhead.

Subjects will be complete an online informed consent form before taking the pretest.

23. Waiver of Consent Documentation and/or Procedure

Waiver of consent documentation: An IRB may waive the requirement for the investigator to obtain a signed consent form for some or all subjects if certain conditions are met and if sufficient justification is provided.

Waiver of Consent Procedure: An IRB may approve a consent procedure which does not include, or which alters, some or all of the elements of informed consent set forth in this section, or waive the requirements to obtain informed consent subjects if certain conditions are met and if sufficient justification is provided.

If waiver(s) is being requested provide brief explanation below of request for waiver(s) AND attach completed waiver form. For more details and downloadable forms, go to:
<http://www.research.uncc.edu/comp/human.cfm>

	YES	NO
Waiver or Alteration of Consent Procedure: <i>Complete appropriate Waiver form and submit with protocol application.</i>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Requesting waiver of some elements of consent?	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Requesting waiver of consent entirely?	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Waiver of Consent Documentation: <i>Complete appropriate Waiver form and submit with protocol application.</i>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Explanation:

Submission Reminders

Submit two (2) signed copies of the entire IRB Approval Application to:

Cat Runden or Dixie Airey, Office of Research Services, 3rd Flr., Cameron Hall.

Have you included the following items?

- ☐ Informed Consent document on appropriate UNC Charlotte letterhead (parental and assent if applicable)
- ☐ Surveys
- ☐ Questionnaires
- ☐ Psychometric Testing Instruments
- ☐ Interview and focus group questions
- ☐ Assessments
- ☐ Pre-Test, Post-Test
- ☐ Inventories, or scales that will be completed by participants
- ☐ Recruitment scripts (email, telephone, verbal announcements) & Flyers
- ☐ Request for Waiver documents
- ☐ Letter(s) of permission/cooperation to recruit participants from and/or conduct the research project from school(s), organization(s) or any off-campus location.
- ☐ Did you include the tutorial completion date for you and ALL of your co-investigators, responsible faculty, research assistants, etc?
- ☐ Grant proposal methodology section, if applicable

Involvement of co-investigators from other institutions:

Efforts to determine the need for IRB approval from the co-investigator's institution/organization must be documented. This documentation may be submitted along with the signed Investigator Agreement which can be found at: **<http://www.research.uncc.edu/Comp/human.cfm>**.

For additional assistance, call Cat Runden at (704) 687-3309 or Dixie Airey at (704) 687-3311.

8.10.2 - Informed Consent

Informed Consent for

Game2Learn: Teaching introductory programming using games

Project Purpose

You are invited to participate in a research study entitled Game2Learn: Teaching introductory programming using games. The vision for our project is to create game to teach introductory computer programming skills. This innovative learning environment uses a familiar, concrete, and engaging metaphor to encourage student learning. In this study, we are comparing a Game2Learn game with a traditional homework assignment to determine the effectiveness of learning to program using games.

Investigators

Dr. Tiffany Barnes, Amanda Chaffin (MS student), Katelyn Doran (Undergrad) Computer Science Department, UNC Charlotte

Eligibility

You may participate if you are currently enrolled in a course in computer programming (ITCS2214/2215) and can speak English.

Overall Description of Participation

You will first be given a demographic survey. Once you have completed the survey, you will then be asked to start the game. Inside the game, we ask you to take a quick pretest that asks 5 multiple choice questions. Once you have finished the pretest, you then play the game until you reach the end. After receiving your score for playing, you are then asked to take a posttest. Finally, you will take a survey on your perceptions of the game and the assignment. Your game playing and task performance will be logged.

The entire process should take no more than one hour, however, as we are asking you to write code, please be aware that this may take longer.

Length of Participation

Your participation in this study will take approximately one hour.

Risks and Benefits of Participation

There are no known risks to participation in this study. However, there may be risks which are currently unforeseeable. Agreeing to participate will allow us to use data obtained from this study to improve computer science education. Refusal to participate in this study will not be reported to your instructor and will not affect your grade.

Volunteer Statement

You are a volunteer. The decision to participate in this study is completely up to you. If you decide to be in the study, you may stop at any time. You will not be treated any differently if you decide not to participate in the study or if you stop once you have started.

Confidentiality Statement

Any information about your participation, including your identity, is completely confidential. The following steps will be taken to ensure this confidentiality:

- **Once recorded, your data will be de-identified by assignment of a random identifier to your records.**
- **Association between participant names and identifiers will be kept in locked files and only study staff will be allowed to look at them.**
- **No personally identifiable information will be released or shared.**

Participants should be aware, however, that the experiment is not being run from a "secure" ftp server of the kind typically used to handle credit card transactions, so there is a small possibility that responses could be viewed by unauthorized third parties (e.g., computer hackers). However, the data accessible this way will only be data logging your game performance.

Statement of Fair Treatment and Respect

UNC Charlotte wants to make sure that you are treated in a fair and respectful manner. Contact the university's Research Compliance Office (704-687-3309) if you have questions about how you are treated as a study participant. If you have any questions about the actual project or study, please contact Dr. Tiffany Barnes (704-687-8577, tbarnes2@uncc.edu).

Participant Consent

I have read the information in this consent form. I have had the chance to ask questions about this study, and those questions have been answered to my satisfaction. I am at least 18 years of age, and I agree to participate in this research project. I understand that I may request a copy of this consent form.

If you are 18 years of age or older, understand the statements above, and freely consent to participate in the study, sign and date this form and give it to the moderator..

Legal Signature: _____

Printed Name: _____

Date: _____

PI: _____

8.10.3 – Professor Recruitment Email

Professor <Insert name here>

The Game2Learn group, headed by Dr. Tiffany Barnes, would like to invite your classes to participate in a study to give your students additional instruction in terms of data structures and recursive algorithms. If you agree, we will talk to your class and ask them to participate in our study and will only take a maximum of five minutes, either at the beginning or the end of the class period. The study will take about an hour and will teach the students about depth first search and recursion in a video game.

If you would like to offer this opportunity to your class, please email us with the date and time that we can come recruit students from your class.

Thank you for your time.

Amanda Chaffin

Ph.D. Student

Computer Science Dept.

UNCC

8.10.4 – Recruitment Poster

**Research Study for EleMental: The Recurrence****Call for Study Participants**

The Game2Learn group, headed by Dr. Tiffany Barnes, is looking for individuals to participate in a research study on a video game that teaches recursion programming concepts. This study will explore the effectiveness of using a compiler inside a video game as well as the effectiveness of the teaching methods employed by the game. Your participation in this study includes filling out a qualitative survey, taking a pretest, playing the game, taking a posttest, and taking a final survey. Participation is voluntary and should take no more than one hour.

Eligibility:

English speaking

UNCC computer science student who is taking 2214 or 2215

This study is voluntary and participants may leave at any time.

Contact Information:

Amanda Chaffin

Ph.D. Student

Address: UNCC, Woodward Hall, Room 453

Email: Katla@wulfkub.com

Katelyn Doran

Undergraduate Student

Address: UNCC, Woodward Hall, Room 453

Email: kedoran@uncc.edu

Dr. Tiffany Barnes

Address: UNCC, Woodward Hall, Room 403 E

Email: tbarnes2@uncc.edu

Phone: 704-687-8577



8.10.5 – *EleMental*: The Recurrence’s Game Design (Brief)

Game2Learn -“*EleMental*: The Recurrence”

Designed and Developed by

Amanda Chaffin - Masters Student

Katelyn Doran - Undergraduate

University of North Carolina, Charlotte

Drew Hicks – Undergraduate

Marietta College

Built on the DarkWynter Game Engine

On the XNA Platform

Abstract –

EleMental: The Recurrence, which is built on top of the DarkWynter game engine, is a game which allows students to write code in a C# compiler that interacts with game objects during run time. Because the students are 2214/2215 students (beginning programmers), we will be selecting the variables which run inside our game code; this will prevent mishaps in the engine. For example, we will take a string variable the student writes and use that as the player's name (see Quest 1 for more details). The game is completely inclusive allowing students to enter all of their code within the game environment. Also, because we are using a C# compiler inside the game and the students are only familiar with C++ and Java, we will provide scaffolding code so that the students will learn the coding structure rather than the language.

The Plot – Players play as a college student, Ele, trapped in her own mind by the Nightmares she has regarding your programming courses. In order to help Ele escape, players must collect her Thoughts. To complete this task, players will have to manipulate the environment and the Thoughts with code. Aiding Ele on her journey through her mind is Cera Bellum, or Cera for short. Cera will provide players with advice to help them towards escape.

Entering the Game –

Once the player runs the game, this is the first screen that they see...

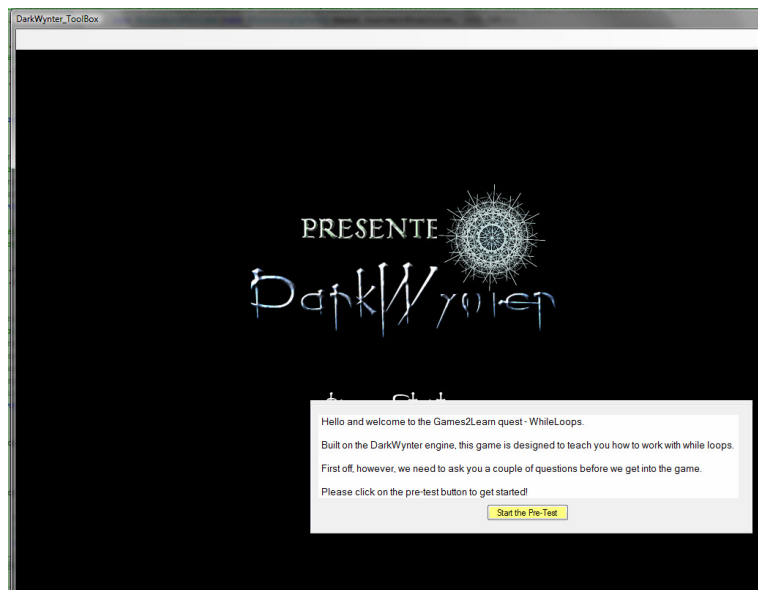


Figure 28: Game Entry Point

Students must complete the pre- test before being allowed to move into the game world. To do so, they select the start pretest button in yellow as shown in Figure 28. Figure 29 shows the

screen that pulls up upon selecting the start pretest button. There is a series of five questions that must be answered before the student is allowed to move into the game world.

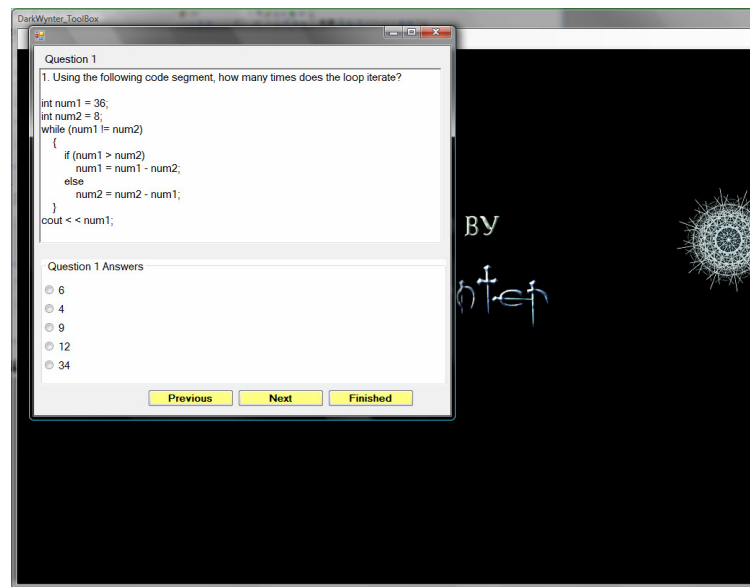


Figure 29: Pretest Survey

Quest 1 – Introduce Yourself and Binary Search Trees

Very simple, this quest is essentially a hello world script, in order to insure that the students are comfortable with programming in the different environment. The students will have scaffolding code that they will modify to print out their name and tell Cera Hello. The scaffolding code...

```
using System;
class Test
{
    public static void MyFunction()
    {
        // Tell us who you are
        string name = "[enter name]";
        Console.WriteLine(name);
        // Tell Cera Hello
        Console.WriteLine("[enter greeting]");
    }
}
```

The students will need to input their name and tell Cera Hello. Once the students compile and run the code, if their changes are correct, the game will signify that by a triumphant sound and

an experience bonus. If the code that is compiled is incorrect, however, they will receive negative feedback in terms of a negative sound, a text message from the mentor, and a loss of experience points. Figure 3 shows the interface the students will use, the instructions, and the code compiled correctly.

Once the code compiles and runs, the student will then receive instructions from Cera to walk down the path that opens. As they walk down the path, dialogue boxes (as shown in Figure 3) pop up to walk the player through the Binary Tree. They can also use the map in the upper right hand corner of the screen (Figure 30) to orient themselves to the world.



Figure 30: Coding Interface and Hello World

As the players walk down the correct paths and get to the leaf nodes in the tree, they can pick up Thoughts along the way, which the game directions tell them to do. If you look at Figure 31, you can see the Thought circled in red (the Thoughts are not circled in the game but they are animated).

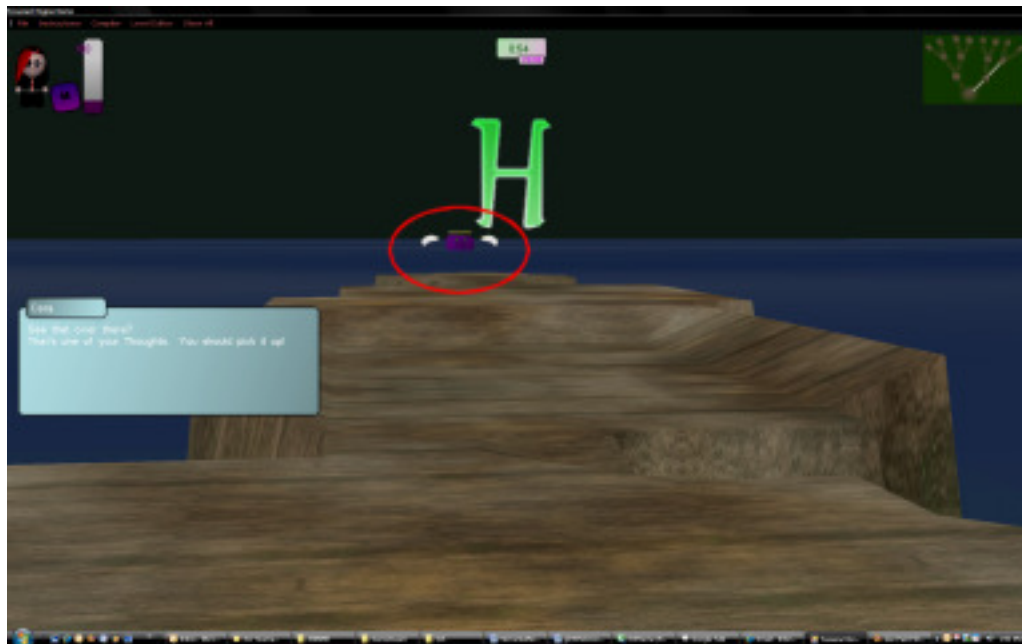


Figure 31: Game Thought

When a student picks up a Thought, a message appears on the screen to let them know they have successfully received the Thought and their Health bar increases (Figure 32).

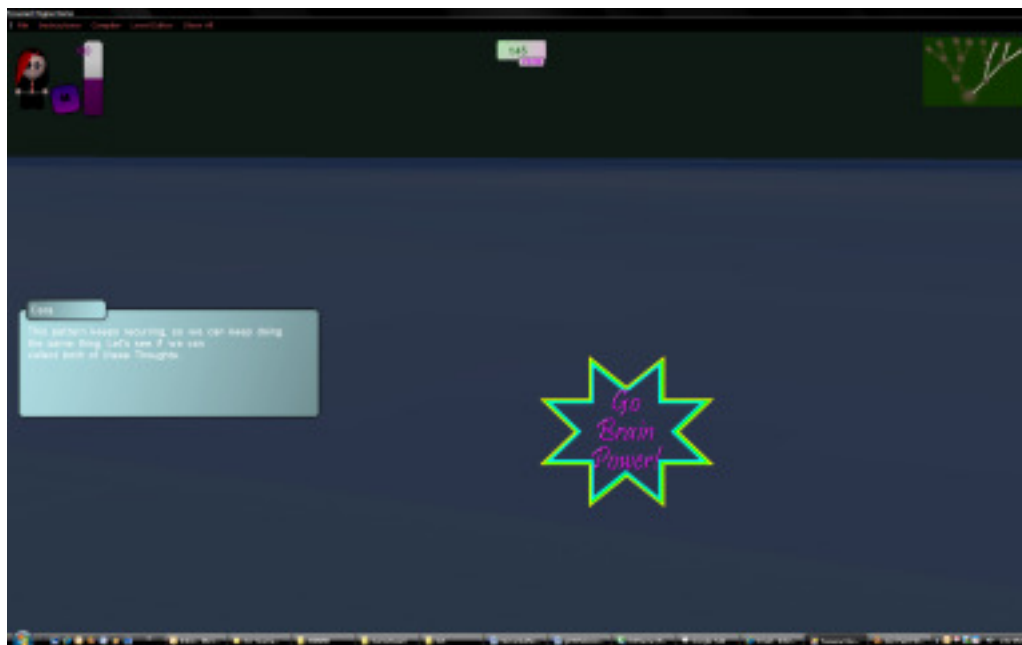


Figure 32: Successful Pickup of a Thought

Once the student successfully completes the walkthrough of a Binary Search on the Tree, they are taken to the second level.

Quest 2 – Non-Recursive Binary Search

Once the student enters the third level, they will immediately notice that the game has changed to an overhead third person view point and that they are not allowed to move about freely in the game world as shown in Figure 33.

For this level, students will be provided with “unravelled” code for a depth-first traversal. Students will compile and run this code. No changes will need to be made by the student to the provided code.

Upon running the code, students will be instructed to watch as a Thought traverses the tree and collects Thoughts on Ele’s behalf. As the Thought moves, Cera will provide clearer instruction of the processes taking place. The dialogue will indicate that the “unravelled” hardcode is not the most efficient way of creating a depth-first traversal. This level will provide a lead-in to the recursive coding in the next level.

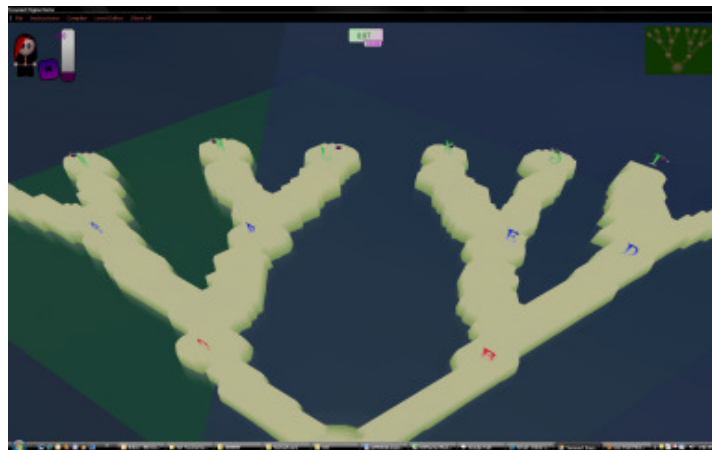


Figure 33: Overhead View

Quest 3 – Code a Recursive Binary Search

The third level is nearly identical to the second level in appearance and gameplay. The code we provide the students with for this part of the game is more complicated as shown by the following scaffolding code:

```
using System;
using DarkWynter.Engine.Compiler;
class Test
{
    public void TreeTraversal()
```

```

{
    Tree myTree = new Tree();
    depthFirstSearch(myTree.root);
}
public void depthFirstSearch(Node node)
{
    Thought.moveTo(node);
    //BASE CASE
    if ((node.returnRight() == null) && (node.returnLeft() == null))
    {
        return;
    }
    //RECURSIVE CASE
    else
    {
        if (node.returnRight() != null)
        {
            depthFirstSearch(node.returnRight());
            Thought.moveTo(node);
        }
        //Use the above if statment
        //to complete the one below
        //this if statement should let
        //you travel to a node's left child
        if ([INSERT_YOUR_CODE] != null)
        {
            [INSERT_YOUR_CODE]
        }
    }
    return;
}
}

```

The students will need to go into the code and change the code where indicated by the comments to perform a search of the tree to the left – the right side and the base case are already coded for them. If they have coded the problem correctly, their code will look like this (changes are highlighted in blue)...

```

using System;
using DarkWynter.Engine.Compiler;
class Test
{
    public void TreeTraversal()
    {
        Tree myTree = new Tree();
    }
}

```

```

    depthFirstSearch(myTree.root);
}
public void depthFirstSearch(Node node)
{
    Thought.moveTo(node);
    //BASE CASE
    if ((node.returnRight() == null) && (node.returnLeft() == null))
    {
        return;
    }
    //RECURSIVE CASE
    else
    {
        if (node.returnRight() != null)
        {
            depthFirstSearch(node.returnRight());
            Thought.moveTo(node);
        }
        //Use the above if statment
        //to complete the one below
        //this if statement should let
        //you travel to a node's left child
        if (node.returnLeft() != null)
        {
            depthFirstSearch(node.returnLeft());
            Thought.moveTo(node);
        }
    }
    return;
}
}

```

Once they run the code, if their code is correct, the game will signify that by a triumphant sound and an XP bonus. If the code that is compiled is incorrect, however, they will receive negative feedback in terms of a negative sound, a text message, and a loss of XP points. Feedback will also be provided so students can correct their errors. Players will have an indefinite number of attempts for the challenge.

If the code is correct, the game will instruct the player to watch the Thought move (Figure 7) through the tree. As the Thought moves through the tree, Cera will provide further information about what is actually occurring with regards to the code. This will enhance the player's understanding of how recursive code runs. Also, the image of a stack will appear on the HUD (Figure 8). This image will demonstrate to students how recursive calls will be pushed into a stack until a value can be returned back down.

For this level, the students will watch the Thought complete a full depth first traversal of the tree. The Thought will collect other Thoughts at the leaves and then return to the root of the tree. Successfully completing this level allows the player, as Ele, to escape the world.

Upon completion, the end screen appears and displays the point total for the round (Figure 34). This point total is based off a rubric of correct code minus a time percentage.

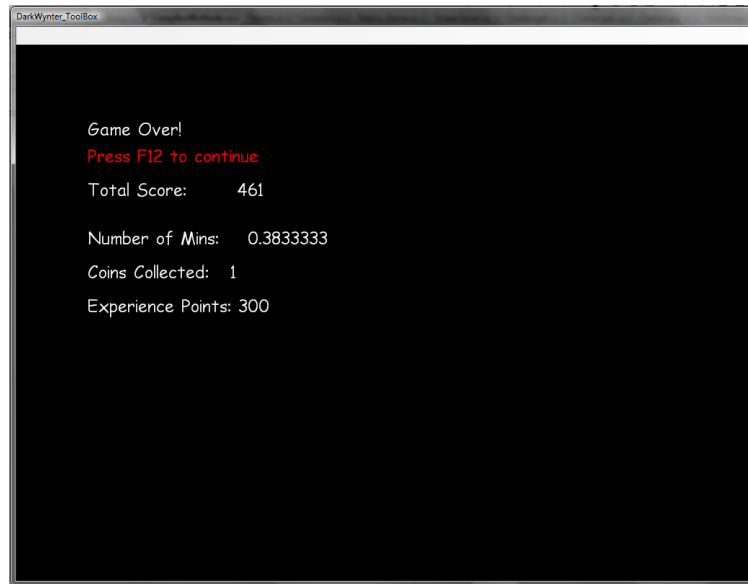


Figure 34: Game Over Score

Once the student presses F12 as requested, the posttest screen is launched as shown in Figure 35.

Question 2

2. Using the following code segment, determine the output of the program.

```
int x = 0;
do
{
    cout << "test" + x + " ";
    x++;
}
while (x != 5);
}
```

Question 2 Answers

☐ test0test1test2test3test4test

☐ test0 test1 test2 test3 test4

☐ 1 2 3 4 5

☐ test test test

Figure 35: Posttest

After the student finishes the posttest, the game writes the pre and posttest questions and answers to a log file as well as the game data. Both files are then FTP'd to a server for storage.