# Experimental Evaluation of Teaching Recursion in a Video Game

Amanda Chaffin, Katelyn Doran, Drew Hicks, and Tiffany Barnes
Department of Computer Science
University of North Carolina at Charlotte
9201 University City Blvd., Charlotte, NC, USA
1-704-687-8577
tbarnes2@uncc.edu

## ABSTRACT

We present Ele*Mental*: The Recurrence, a novel game that provides computer science students the opportunity to write code and perform interactive visualizations 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 visualizations. 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.

## Categories and Subject Descriptors

K.3.2 [**Computers and education**]: Computer and information science education. – computer science education.

## General Terms

Design, Human Factors

## Keywords

Game development, education, motivation, evaluation.

## 1. INTRODUCTION

Even as employment opportunities for computer scientists rise, the number of students enrolling and graduating from college with computer science (CS) degrees is on a steady decline [Vesgo 2007]. Most of the attrition in the CS degree happens early on, typically in the first two CS courses, 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].

Many researchers have discussed the possible benefits for using games in education and we believe they may be particularly helpful in improving computer science education. Games provide a way to create and share educational content while also making students feel their computing education is more relevant. We hope to leverage the increased motivation games inherently provide to increase student attitudes toward computing and help them learn at the same time. 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 [Losh 2006; Jenkins 2006]. In computer science, building video games provides 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].

## 2. BACKGROUND

Games are increasingly recognized for their inherent motivation, as inspiration for improving educational applications [Barnes et al. 2008; Garris et al. 2002; Parberry et al. 2005]. However, there is little consensus on what makes for an effective instructional game [Gee 1999]. Lepper and Malone have investigated the most important factors in making educational games fun, listing as most game design books do the importance of challenge, or the balance between ease and difficulty, in engaging learners in games, but highlight the need to design in activities that help learners address and revise their misconceptions [1987]. Garris, Ahlers, and Driskell [2002] have classified the factors that are important to games' effectiveness for learning, and have identified motivation to play and play again as a key feature of the best instructional games. Their framework reveals that is important to weave game engagement in with the types of feedback most useful in learning [Garris et al. 2002]. We have therefore striven to design educational games that balance play and learning time, make strong ties between in-game motivation and learning outcomes, and that can be used as learning tools in a standard introductory computing course.

Games have long been used to engage students in learning computer science. Becker [2001] found that programming Minesweeper and Asteroids in the first two computer science courses can help students better understand object inheritance [Becker, 2001]. Bayliss and Strout teach introductory computer science using games at the Rochester Institute of Technology (RIT), and have found improved performance and decreased feelings of intimidation by peers [Bayliss, Stout, 2006].
Games are also used as advanced projects for students in capstone projects. At the University of North Texas, most computer science majors complete the successful capstone course, where they work in teams to create games of their own choosing using


Sandbox 2009, New Orleans, Louisiana, August 4 – 6, 2009.
© 2009 ACM 978-1-60558-514-7/09/0008 $10.00

realistic tools and practices [Parberry et al, 2005]. This work encouraged us to include real programming tasks in our games.

Developed at RIT, MUPPETS (Multi-User Programming Pedagogy for Enhancing Traditional Study) is a game where students develop and interact with visible 3D objects in the game world [Bierre & Phelps, 2004]. Using Java, students can edit existing code, create new code, compile and run their code and have direct feedback in the form of compilation errors or, in the case of the code being correct, have the changes appear in the game. In the introductory course, students program their robots to compete in a class-wide battle. The MUPPETS approach is very similar to our own, but we build in more scaffolding and instruction into our games.

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 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 feel like they can solve problems than those who took traditional courses (Mullins, et al. 2008).

Scratch, a visual programming language, was created at the Michigan Institute of Technology (MIT) by the Lifelong Kindergarten Group in collaboration with UCLA and teaches students how to program in a visual learning environment. Students can create custom scripts, with some advanced programming concepts (sequential execution, threading, user interaction, conditionals, etc.) as well as more basic programming concepts (variables, print statements, etc). 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 they would not quite realize what they are doing with the program (Maloney, et al 2008).

At UNC Charlotte, undergraduate students have built and evaluated several Game2Learn games for teaching computing. Saving Sera [built in RPG Maker by Enterbrain Corporation] and The Catacombs [built in Neverwinter Nights by Bioware Corporation] are role-playing games (RPGs) designed to teach control logic such as if-then statements and for loops [Barnes 2007, 2008]. User studies of these games showed positive learning gains, and after we made some changes to add more feedback, positive attitudes about using such games as homework. However, when we used these games with CS1 students, we found that they were not focused enough and caused some students' confusion. Therefore, we now design our games to concentrate most of the gameplay on learning rather than world exploration. Squee and The Tournament, two games for teaching functions, have also shown positive learning gains, though this work is currently unpublished. From Squee, we learned that a complex interface can overshadow the benefits of a good metaphor, and with The Tournament, we learned that a three-dimensional (3D) RPG with interactive dialog can make the presentation of learning material more engaging than a lecture.

Wu's Castle is a 2D RPG where players must build armies of snowmen to escape a mirror universe [Eagle 2008]. Of course, students learn to use arrays and for loops to construct these armies. We have conducted extensive studies with Wu's Castle and have shown playing the game before doing a programming assignment to learn loops and arrays can increase overall learning [Eagle 2008, 2009], and we now use this game in UNC Charlotte's introductory programming course. From this game, we have learned the importance of full logging of game interactions and learning outcomes, which can indicate what parts of the game are working best. We also found that interactive instruction with small steps can help lead the players through the learning process while still keeping the players engaged and feeling as if they are "playing a game".

## 3. GAME2LEARN DESIGN & ENGINE

Designing educational games requires a different focus than general game design; otherwise, we may fall into the trap of designing fun games with no learning value [Barnes 2007]. During the course of the Game2Learn project, we discovered it is best for the game designers to select first the target concept they want to teach. Then the designers create the code that best illustrates the target concept. 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 design and game engine choice to best match the target computing concept and metaphor.

After the overall game concept is defined, the designers then tailor the game instructions to the students to support 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, that is, 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.

We used this Game2Learn methodology to design Ele*Mental*: 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, combined with the capability to write, compile, and run a program within the game environment.

Ele*Mental* was created using a game engine called DarkWynter that was created by a team of students in UNC Charlotte's Game Design and Development program [DarkWynter Studio]. Unlike commercial game engines, the DarkWynter engine's code base is open source for academic purposes. The engine is coded using XNA's Game Studio 2.0, and integration with the C#'s Code Dom allows us flexibility to create games that could compile any .Net language [Microsoft Corporation].

The DarkWynter engine allows for real-time terrain modification that we planned leveraged for interactive visualization of depth

first search (DFS) in a binary tree. With an obvious data structure (a tree), a standard algorithm to code (DFS), and the majority of the implementation needed for gameplay built into the engine (real time terrain modification), we were quickly able to finalize the code we wanted players to produce.

Since the game allows coding in C#, but our students learn C++ or Java, we designed the game code to be relatively free of C# quirks. To avoid teaching students more about C# than necessary, we wrote scaffolding code for all the coding challenges. Finally, we created a parsing method to check both the students' code and their output to insure that their code matched what we wanted them to produce. After completing the parsing section, we then turned our attention to the design of the game.

## 4. ELEMENTAL GAME DESIGN

The overall design of the game involves completing three programming puzzles, helped by Ele, a programmable avatar shown in Figure 1, Thoughts shown in Figure 2 for visualizing data to collect, and Cera, an in-game mentor, who instructs students as they progress through the game, with dialog as shown in Figure 3.

**Figure 1: Ele Icon**

**Figure 2: Thought Icon**

In the game, students first take a brief pretest to determine their understanding of recursion. Then students complete a basic "hello world" program to get used to the compiler interface. Then students walk their character using depth first search traversal to collect Thoughts from the leaves of a binary tree. In level 2, students must correctly code the traversal for the left side of DFS. After their code is written, Ele walks through the tree using the student code while Cera explains what the code is doing. 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, 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.

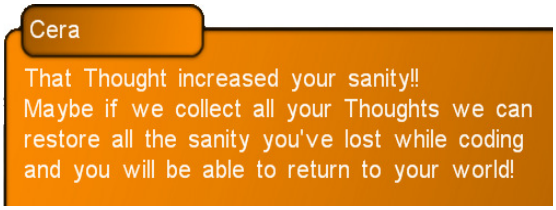


**Figure 3: Cera's Dialogue Box**

### 4.1 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, such as that in Table 1, as well as its own set of instructions.

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 must code their way back to normality. We also explain the only real difference between C# and C++ or Java code we are going to ask them to write, is 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.

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 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, as shown in the scaffold code given in Table 1. In level 3, the students write all the code for the recursive portion of the DFS.

**Table 1: Level 2 DFS scaffold code**

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

     // Check for Base Case
     if ((node.returnRight() == null) &&
         (node.returnLeft() == null))
     {
        return;
     }
     else      // Recurisve calls
     {
        // Travel to node's right child
        if (node.returnRight() != null)
        {

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

If, at any time during their coding, the students compile incorrect code, they receive the compiler error message, including which line is incorrect. 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 indicating where to correct the error.

### 4.2 Traversal and Instruction

After the student finishes the code portion in each level, they walk through the tree in DFS order themselves or watch their programmed AI do so. In the first level, the students walk through the tree as Cera provides 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 displays only the tree structure and the student's progress.

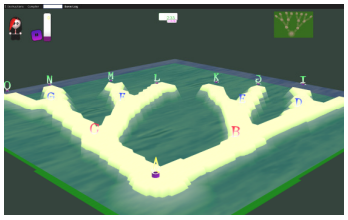

**Figure 4: Level 2 AI walkthrough**

In the second level, after the students write the left portion of the DFS, they watch the programmed AI traverse the tree from an overhead view shown in Figure 4. As the traversal proceeds, dialogue is shown explaining how recursive calls are made to direct the character, as in Table 2.

**Table 2: 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 level 3, students traverse the tree once again. However, this time; we added a stack and 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 5.


**Figure 5: Visualization of a DFS stack showing each node**

Table 3 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.

**Table 3: 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 can't. Eric 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."
- "So now, your calls have returned a value to you and have popped off. You can continue your calls by calling Cody(C)."

## 5. METHOD

We performed an exploratory evaluation of the prototype with 43 students who were enrolled in or had already completed Data Structures and Algorithms between October 2008-February 2009. The purpose of the study was to gather formative feedback and determine the impact and feasibility of using such games for homework. Participants signed an informed consent form, took a demographic survey and a pre-test of recursion-related computing concepts, played about 40 minutes of the game, took a post-test, 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 post-test is similar to the pre-test, with the numbers and variable names changed in each question (so they are isomorphs but not identical). Each session took about 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 coding 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 pre-test/post-test results to categorize responses to see if different groups responded differently to the game.

## 6. RESULTS & DISCUSSION

The study was conducted with 43 participants. The first twenty-seven took only a post-test, while the final 16 participants took both the pre-test and post-test. In this discussion we focus on the pre- and post-test group. Of these 16, 13 were 18-25 years old and 3 were 25-30. Thirteen 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 student. Fourteen were in computer science-related majors, 1 was in computer engineering, and 1 was in graphic design with a minor in computer science. The students were also asked what programming languages that they had used previously. As Java and C++ are taught in our first series of programming courses at UNCC, it comes as no surprise that these were the dominant languages among the students.

We also asked the students what types of games they like to play when they have the time. Interestingly enough for our project, 8 of the students like first person shooters, which is what the DarkWynter game engine was originally, and 8 of the students like RPGs, which is what the engine has become.
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.

### 6.1 Quantitative Results

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 2. We note that all scores increased except for Q3. We suspect this may be a result of students assuming the pre- and posttest questions were identical. 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 there is a significant improvement in the posttest scores compared to the pretest scores. The effect size, using Cohen's d, is .78, which is a large effect size, with an $\alpha$ error rate of .05 and a power of .95. On average, the posttest scores were approximately .78 standard deviations higher than the pretest.

**Table 2: 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 5 shows the pretest and posttest results, with participants, ordered by increasing post-test scores, on the x axis and scores (out of 5 points) on the y. Each participant has two columns in the chart – the light blue column representing their pretest score and the purple representing their posttest score.

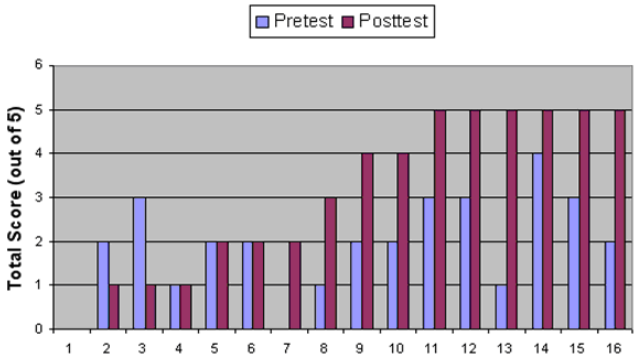


**Figure 4: Pre and Posttest Scores per Participant**

Since some students did not take the 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 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 pretest before the posttest.

Table 3 shows the averages and standard deviations over the game logs for each student in group P, which include the overall time spent (minutes: seconds) on the game, as well as the times and the number of tries students took on each challenge. 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 11 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 3: 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 |

Figure 6 shows the results of our analysis of the 11 complete log files for a correlation between time spent playing the game and code attempts with their test scores. The x-axis lists the total time time for each student, with three bars for each student whose

heights represent their pre-(blue) and posttest (purple) scores and number of attempts (yellow). There was not a significant interaction between time and attempts. However, there is 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.

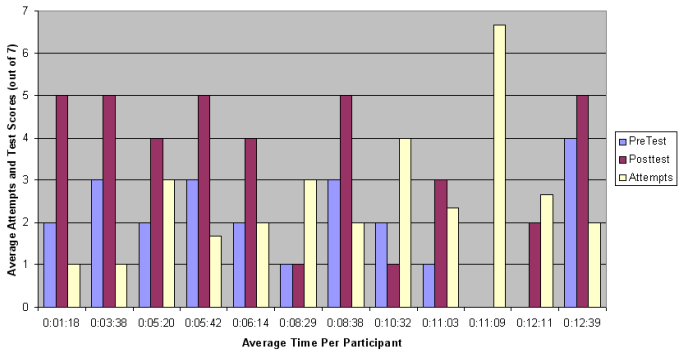


**Figure 5: Time in Game and Attempts vs. Test Scores; Participants Ordered by Time Spent**

## 6.2 Survey Results

While we only have valid pre- and posttests for 16 students, 42 students gave us survey responses after playing the game. We have averaged the results from the survey questions, where one equals strongly disagree and five equals strongly agree. Out of the 42 students, 34 strongly agreed or agreed that they enjoyed playing the game. Thirty-five respondents agreed that they enjoyed creating and compiling their own code inside the game (88% response). Thirty-four of the students agreed the game was helpful in learning computer science concepts and 32 students thought the game has a good balance between play and quest time. Thirty-four students 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 that they guessed at the solutions.

Table 4 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...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."

**Table 4: Favorite aspects of Ele*Mental***

| 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) |

We categorized 42 responses to the item "What improvements would you suggest for Ele*Mental*?" as shown in Table 5. Several students suggested we include player orientation in the mini map for navigation, improve the instruction language, and to fix bugs, such as those that allow students fall off the bridges. The three students who suggested we add audio (though it is included) did not use headphones. Two students asked for more levels with harder code quests and more game play.

**Table 5: 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 game 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% prefered traditional assignments and the remaining students were neutral. One student wrote, "I believe game assignments are better than traditional ones because [they] make you think more outside the box." Another student expressed "Game 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, "Game 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."

## 7. CONCLUSIONS & FUTURE WORK

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 student game programmers included novices, we also had to ensure 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.

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. In order to ensure players wrote the expected code, student developers had to learn how .Net compiler works and how to parse code. The process improved our knowledge of recursion, compilers, abstraction, polymorphism, and designing complex architectures to handle game interactions. By using the agile cycle of development, we learned how to iterate rapidly through the software cycle efficiently and with good communication. Perhaps most importantly, student developers learned good code is no substitute for good design.

Our study results show students achieved statistically significant learning gains when they played Ele*Mental*: The Recurrence.

Although the sample size 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 Ele*Mental*: The Recurrence implements effective teaching strategies in a game environment. With the good use of visualization, frequent feedback, and an in-game compiler that students can write their own code to work in our game, this game demonstrates many characteristics we feel are important for educational games for computing. Most of the students 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.

Based on our findings in this study, we plan several improvements for EleMental. 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. In all levels, we will also include a visualization of how the student's programmed traversal would work, even when it is incorrect. We may also remove the first-person perspective for walking through the binary tree, since a top-down view seems to be most beneficial.

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 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.

## 8. ACKNOWLEDGMENTS

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.

## 9. REFERENCES

ACM/IEEE-CS JOINT CURRICULUM TASK FORCE. Computing Curricula 2001. Accessed February 13, 2008. http://www.acm.org/education/curric_vols/cc2001.pdf

ASPRAY, W., AND BERNAT, A. Recruitment and retention of underrepresented minority graduate students in computer science: Report of a workshop, March 4-5, 2000. Washington, DC: Computing Research Association.

BARNES, T., E. POWELL, A. CHAFFIN, H. LIPFORD. Game2Learn: Improving the engagement and motivation of CS1 students. ACM GDCSE 2008.

BARNES, T., H. RICHTER, E. POWELL, A. CHAFFIN, A. GODWIN. (2007). Game2Learn: Building CS1 learning games for retention, Proc. *ACM Conference on Innovation and Technology in Computer Science Education* (ITiCSE2007), Dundee, Scotland, June 25-27, 2007.

BAYLISS, J., AND S. STROUT. Games as a "flavor" of CS1. In SIGCSE 2006. ACM Press, New York, NY, 500-504.

BEAUBOUEF, T. AND J. MASON. Why the high attrition rate for computer science students: some thoughts and observations. SIGCSE Bull. 37, 2 (Jun. 2005), 103-106.

BECKER, K. 2001. Teaching with games: The Minesweeper and Asteroids experience. The Journal of Computing in Small Colleges Vol. 17, No. 2, 2001, 22-32.

BIOWARE CORPORATION. Aurora Neverwinter Toolset. Accessed Jan. 2, 2008. http://nwn.bioware.com/builders/

DARKWYNTER STUDIO. The DarkWynter 3D Game Engine. Accessed March 27, 2008. http://darkwynter.com/

EAGLE, M., T. BARNES. Experimental evaluation of an educational game for improved learning in introductory computing. ACM SIGCSE 2009, Chattanooga, TN, March 3-8, 2009.

ENTERBRAIN CORPORATION. RPGMaker XP. Accessed Jan. 2, 2008. http://www.enterbrain.co.jp/tkool/RPG_XP/eng/

GARRIS, AHLERS, AND DRISKELL. Games, motivation, and learning: a research and practice model. *Simulation and Gaming*, Vol. 33, No. 4, 2002, 441-467.

GEE, J. P. What video games have to teach us about learning and literacy. *Compute Entertain of Educational Technology*. 30, 4 (1999), 311-321.

JENKINS, HENRY, ERIC KLOPFER, KURT SQUIRE, AND PHILIP TAN. Entering the Education Arcade. *Source Computers in Entertainment*, Volume 1, Issue 1 (Oct.. (October 2003), 20.

LEPPER, M. R., AND MALONE, TH. W. 1987. Intrinsic motivation and instructional effectiveness in computer-based education. In R. E. Snow and M. J. Farr (Eds.), Aptitude, learning, and instruction: Vol. 3. *Conative and affective process analyses* (pp. 255-286). Hillsdale, NJ: Lawrence Erlbaum.

LOSH, ELIZABETH. In Country with Tactical Iraqi: Trust, Identity, and Language Learning in a Military Video Game. *Virtualpolitik*. 2006. August 3, 2006. http://virtualpolitik.org/DAC2005.pdf

MALONEY, J. H., K PEPPLER., Y KAFAI, 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

MULLINS, P, WHITFIELD, D, AND CONLON, M. 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).

PARBERRY, IAN, MAX B. KAZEMZADEH, TIMOTHY RODEN. The art and science of game programming, Proc. 37th SIGCSE *Technical Symposium on Computer science education*, March 03-05, 2006, Houston, Texas, USA.

BIERRE , KEVIN J. AND ANDREW M. PHELPS. The use of MUPPETS in an introductory java programming course, SIGITE 2004, October 28-30, 2004, Salt Lake City, UT, USA.

POLACK-WAHL, J. AND ANEWALT, K. Undergraduate research: Learning strategies and undergraduate research. SIGCSE 2006: 209 – 213.

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

VESGO, 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.

ZWEBEN, S. 2006-2007 Taulbee Survey. Computing Research News, 20, 3 (May 2008).