Evaluation of a Game-based Lab Assignment

Michael Eagle
University of North Carolina at Charlotte
Computer Science Department
9201 University City Blvd.
1-704-687-8577
mjeagle@uncc.edu

Tiffany Barnes
University of North Carolina at Charlotte
Computer Science Department
9201 University City Blvd.
1-704-687-8577
tbarnes2@uncc.edu

ABSTRACT

We have developed a learning game to teach loops, nested loops, and arrays using scaffolding and interactive visualization. We compare the game to a traditional programming assignment in an introductory computing laboratory. In our study, 17 Introduction to Computer Science labs were randomly assigned to play the learning game first and half to write a program first. Our results show that students playing the learning game first learn more, and that students prefer the game assignment to the traditional assignment. These results suggest that incorporation of game-based assignments is beneficial to student learning.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education - *computer science education.*
K.8.0 [**Personal Computing**]: General – games.

General Terms

Design, Experimentation, Human Factors

Keywords

CS1-2 education, iteration, arrays, educational games, retention

1. INTRODUCTION

The field of computing is in a crisis – with rising demand for talented professionals but declining enrollments [15], computing educators and professionals alike are looking to find ways to appeal to today's students. Most students leave after taking their first or second computing course, with attrition rates as high as 30—40% [4]. Games have increasingly been used to improve interest and retention in undergraduate programs in computer science through assignments, curricula, and undergraduate research [14], e.g. [2] [8] [3]. However, many curricular approaches to using games emphasize building games, and only a few of these approaches do so in introductory computing courses (e.g. [3]). The Game2Learn project at UNC Charlotte is developing an approach to using game-based learning in introductory computing courses; the project provides students with games experience in introductory courses, without burdening instructors with designing and developing game content.

ICFDG 2009, April 26–30, 2009, Orlando, FL, USA.

In our two-pronged approach described in [2], we engage advanced students to develop educational games for introductory computing students. Introductory students then *play* these games and learn in a more enjoyable, relevant context that helps make abstract concepts more concrete. This project is successful in supporting undergraduate retention and recruiting into computing graduate programs [2], while also revealing some important aspects of educational games through user studies [8].

Although the potential benefits of games in education have been examined relative to their qualitative elements (e.g. challenge, and curiosity [9] [13] [12]), and their social contexts [10], the study of games has lacked a coherent research paradigm [13], and there is a need for controlled studies to reveal the relative importance of game characteristics for learning. In this paper, we present the third in a series of a controlled user studies using *Wu's Castle*, a student-developed game for teaching arrays and loops. Wu's Castle differs from other game-related introductory programming instruction such as that offered by Alice [5] or Scratch.mit.edu, which are visual programming environments that do not provide learning tasks to players. In this, Alice and Scratch are more constructivist learning environments than Wu's Castle as a whole, but our game does offer a sandbox mode to allow for player exploration.

Loops and arrays are two of the top three programming topics reported to be difficult for novice students, according to a survey of computing educators [6]. Debugging and testing of code were also noted as important and difficult [6]. Compiler errors often appear cryptic to novice students, and logic errors, where the code compiles but produces incorrect results, can be even more bewildering. Beauboeuf cites poorly planned labs, where teachers debug the students' code, and a lack of practice coding and debugging with meaningful feedback, as a cause for attrition in computer science courses [4]. There is clearly a need for educational systems that can assist students in anticipating and finding errors, offer opportunities for practice, and provide individualized feedback.

The game, *Wu's Castle*, was developed by an undergraduate student in the *Game2Learn* as described in [2]. In *Wu's Castle,* the player interactively constructs C++ code to solve in-game problems. The player can interactively visualize code stepwise as it executes and easily identify and correct logic errors. We hypothesized that (1) playing the game would improve learning over traditional programming assignments and that (2) playing the learning game before writing a program would improve learning. We also hypothesized that students would complete the programming assignments faster after having played the game. We used a crossover or "switching replications" experimental design to compare the effects of the learning game with a

traditional programming assignment. We randomly assigned 17 Introduction to Computer Science labs into two groups with alternating orders of the assignment (game-first or game-second). In the first phase of the experiment students in the game-first group achieved significantly higher learning gains. In the second phase, the game-second group closed the gap with the game-first group, although scores were still lower. The game-first group learned significantly more than the game-second group. However, we did not find any differences in programming time. Our findings suggest that playing a learning game before writing a program has a positive impact on learning, but may not make students immediately faster at completing programming assignments.

2. THE GAME & ASSIGNMENT

As a senior project, an undergraduate student developed *Wu's Castle,* a 2-dimensional role-playing game made in RPG Maker XP using the rapid-prototyping technique described in [2]. A pilot study of *Wu's Castle* showed significant effects on learning for CS1 students who played the game for extra credit [7]. A follow-up study compared *Wu's Castle* to traditional programming assignments and found significantly higher learning gains for students assigned to do *Wu's Castle* for homework [8]. In [7] and [8] students were given a week to downloaded *Wu's Castle* and play it from home. In [8] students had an additional week to complete a programming assignment designed to simulate the learning exercises in the game. In [8] we used a strong switching replications experimental design comparing students who played the game one week and wrote a very similar program the next week, with the Control group who wrote the program first and then played the game. We concluded from this study that playing the game first might help students build a better conceptual understanding of for loops, nested loops, and arrays, facilitating increased learning over time.

We hypothesized that improved learning gains from playing the game first would also translate to faster programming times, which we could not measure in the previous study. In the current study, we ran a similar experiment, but this time in all sections of our newly designed CS1 course, that includes a required lab. In the current work, students complete *Wu's Castle* and a traditional programming assignment in a single lab session lasting two hours and 45 minutes. The course instructor designed the programming assignment independently.

2.1 The game

In this section, we will give a brief description of *Wu's Castle*; a more extensive description is given in [7] [8]. *Wu's Castle* features a light-hearted story line; Arshes, the player character, falls into a mirror universe and must build an army of snowmen to escape. However, in order to build the snowmen, the player must set loop parameters, select proper code bodies, and walk through code execution visualizations.

The player can walk around the world and interact with several non-player characters (NPCs). Some of the characters offer instruction, while others offer challenges. Figure 1 shows an example of the player interacting with a NPC.

The in-game instruction is always interactive; the player is never required to read long segments of text. To complete the instructional portions of the game, the player is often required to read and analyze code. After the NPC demonstrates a program, the player is required to change the parameters and see how the program behavior changes.

The challenges students encounter in the game start with setting simple parameters to fill a single array (such as those in (for i=0; i<=19; i=i+1)) and progress to more challenging patterns (such as outfitting odd-numbered snowmen with arms, or editing specific sub-sections of an array.) After students have mastered the one dimensional array portion, they move on to two-dimensional arrays, where the player must use nested for-loops to edit the rows can columns.

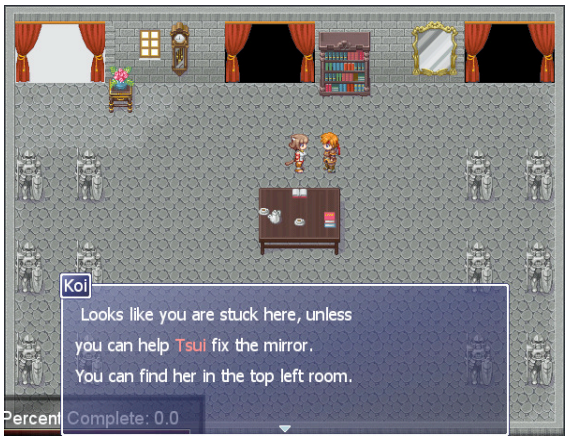

Figure 1: A player character interacting with a NPC.

2.1.1 Core game mechanic

The game was designed to emphasize conceptual learning and to make the abstract representations of arrays and for loops more concrete for introductory students. There are four main levels to *Wu's Castle*, built to increase in complexity. Players experience two types of game play: loop walk-through and array manipulation. In Level 1, players "walk" Arshes through a circular garden path, a physical representation of a for-loop. Using the arrow keys, players walk Arshes along the path, and as he passes a chest (representing a line of code), the variables used in the loop are displayed and updated on the screen. Players cannot move off the loop until the loop ending condition is met. Although this may seem simplistic, many students have incomplete and incorrect understanding of loops that result in incorrect predictions of loop behavior, especially when loops are nested. Figure 2 shows a screenshot of Level 1 showing the code chests to (1) initialize variables, (2) enter the loop, (3) output x, (4) divide x by two.

In Level 2, players first create a full 1-dimensional array of snowmen by setting the start, end, and increment parameters for a for-loop. In Figure 3, the player has set the initial condition to zero and is about to set the stopping condition to 19. After inputting all of the parameters, the player selects the correct type of snowman to enter into the array as shown in Figure 4. Next, players modify subsets of the array, such as every other snowman, with arms and hats. The mission is always displayed at the top of the screen as seen in Figure 3, so that the student will not forget their current task.

In Level 3, the player must walk though a nested loop. During this stage, the player must answer several multiple-choice questions about the code; such as, "which variable is responsible for the outer loop." The forth level is similar to the second, the

difference being that the player must now edit two for-loops in order to edit the two-dimensional array.

The game offers a chance to edit the array without mission restrictions at the end of each level. This allows students a chance to play with the parameters of the for-loops without any in-game penalties for answering the questions wrong.

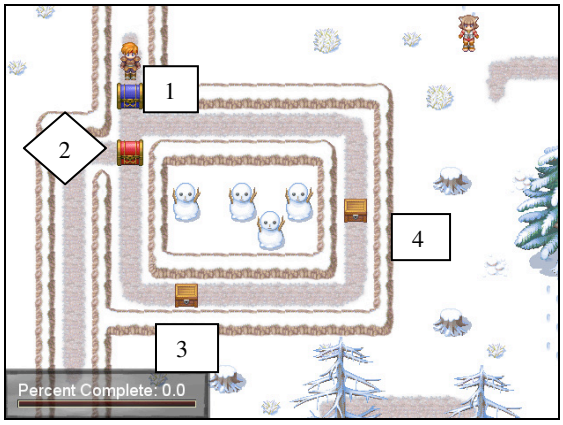

Figure 2: A player walks through a loop visualization.

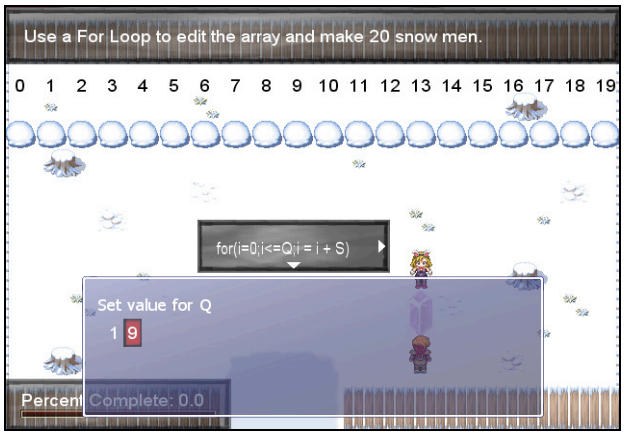

Figure 3: A player sets the stopping condition of the for-loop.

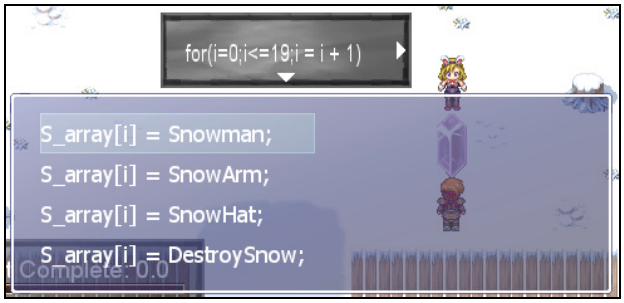

Figure 4: A player selects the snowman type for the loop body.

2.1.2 Feedback

Since feedback can positively affect student attitudes and performance in learning games [2], *Wu's Castle* provides sound and image feedback after every player interaction; Figure 5 shows a positive example. Infinite loops cause the NPCs to faint and out-of-bounds errors cause explosions that shake the screen. Players also receive experience for each mission completed, and earn bonuses when they complete missions in 1-2 tries.

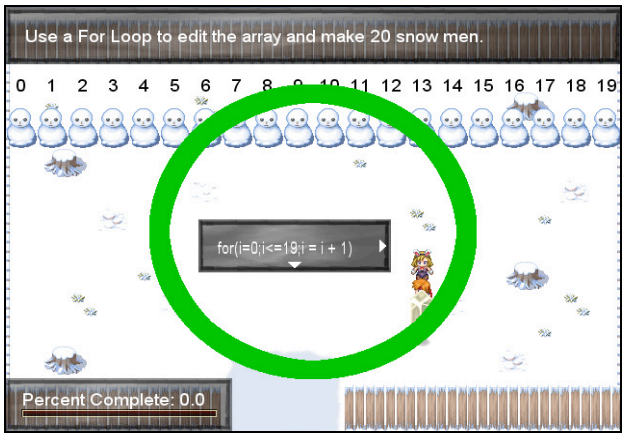

Figure 5: After completing the mission successfully, a green circle appears accompanied by a positive sound effect

2.2 Traditional programming assignment

For the traditional programming assignment shown in Table 1, students were asked to create two small programs. The first requires a single loop, and the second requires a more complicated nested-loop solution.

Table 1: Traditional programming assignment

Write a program that uses a loop to display "X bottles of beer on the wall!" repeatedly from 99 down to 0. Your output should look like the following: 99 bottles of beer on the wall! 98 bottles of beer on the wall! . . (and so forth) . 0 bottles of beer on the wall!
Write a program that uses nested loops to display the multiplication tables for the numbers 1-9. The outer loop should move through the rows and the inner loop should print each column. Your output should look like the following: 1 2 3 4 5 6 7 8 9 2 4 6 8 10 12 14 16 . . (and so forth) . 9 18 27 36 45 54 63 72 81

3. METHOD

We used a crossover, "switching replications" experimental design. We took three waves of measurement, one before any treatments, and one after each treatment. This experimental design removes social threats to validity, allowing all students equal access to the game assignment; it also allows us to study the ordering effects of the assignments. The study was conducted in the first semester course in a three-course introductory computer science sequence; we refer to it as CS1 here but it includes fewer topics than the two-course CS1 courses described in CC2001 [1]. The CS1 course consists of a lecture and a required separate lab.

There are two sections of the lecture taught by two instructors. There are 17 different lab sections, taught by 10 teaching assistants (TAs), seven of whom teach two sections each. Each lab section, with students from both lectures, meets once a week for two hours and 45 minutes. The labs are set up two have one computer for each pair of students.

Table 2: Sample pre- and post-test questions

Q1	Given this declaration: int Num[7]={1,9,7,3,11,5,13}; What are the values of the expressions below? Num[2] Num[4] Num[7] Num[2]+Num[5] Num[0]

Q2	Given the following code segment:
	int x=7; int Num[3][2];
	for (int j = 0; j <= 2 ; j++)
	for (int k = 0; k <= 1 ; k++)
	{ Num[j][k] = x; x++; }
	What are the values in the last row of Num?

Q3	Given the following code segment:
	int Num[100];
	for (int k = 0; k <= 99; k++)
	Num[k] = 1; // Line A
	if (k > 10)
	break;
	How many times will Line A be executed?

Q4	9 8 7 6 5 4 3 2 1 0 9 7 5 3 1 6 7 8 9 0 1 3 5 7 9 What are the values for each of the expressions below? Array[3][2] Array[5][4] Array[2][3] Array[2][1] + Array[0][1] Array[1][3] + Array[3][3]
Q5	for(int i = 0; i <= 5; i = i + 2) Array[i] = checked; Array = [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]

Q6	for(int i = 2; i <= 3; i++)
	for (int j = 0; j <= 3; i = i + 2
	array[i][j] = checked;
	
	Array =
	[ ] [ ] [ ] [ ]
	[ ] [ ] [ ] [ ]
	[ ] [ ] [ ] [ ]
	[ ] [ ] [ ] [ ]

Q7	public static int maxPos(int[] y, int first, int last)
	{
	/* Returns the position of the maximum element in the */
	/* array "y", between positions "first" and "last". */
	int bestSoFar = first;
	*** missing code goes here ***
	return bestSoFar;
	}
	The code should search the array from the high subscripts to the low subscripts. Write the missing code to determine "maxPos".

We randomly assigned 17 different CS1 lab sections into one of two groups. TAs who taught two sections taught one control and one experimental group. We attended the weekly TA meeting and instructed them on how to run the study, and provided a web page for students in each lab section with links to study elements in the proper order. Students completed the game assignment and programming assignment in pairs; however, the students worked separately in turns on pre- and post-tests. Students took a pretest, performed Task 1 (in pairs), Posttest1, Task 2 (in pairs), and Posttest2, which included a post-survey. In the Game group, Task 1 is the game, and Task 2 is the programming assignment. We reverse this order for the Control group. There was no time between interventions. We received the three test scores, game logs, and programming assignments from 137 students.

The pre- and post-tests were designed to be isomorphs, with variable values and question order rearranged between the tests. Table 2 contains sample questions. The questions use the guidelines in [11] to test learning at all levels of Bloom's taxonomy. Each question was worth one point for a maximum of score of seven. The pre- and post-tests and surveys were administered online through (www.surveymonkey.com).

4. RESULTS

We took measures at pretest (**P0**), posttest1 (**P1**), and posttest2 (**P2**). The two groups differed only in the order by which they complete the two assignments. We refer to the group that completed the game first as group **Game** and the other as the **Control** group, even though both groups complete both treatments. We submitted the test scores to a 2x3 repeated measures ANOVA with Group (Game and Control) as a between-subjects factor and Test (P0, P1, and P2) as within-subjects factors. All contrasts use two-tailed tests with an alpha of .05.

4.1 Main Effect of Test

An overall analysis of variance for repeated measures showed a significant main effect of Test, F (2, 270) = 90.123, $p < .001$, Partial $\eta^2 = .40$. This indicates a difference occurring between the tests (P0-P2). There was a significant simple effect of Test for Game group F (2, 126) = 94.821, $p < .001$, Partial $\eta^2 = .601$, and a significant simple effect of Test for the Control group, F (2, 144) = 27.163, $p < .001$, Partial $\eta^2 = .274$. All pretest and posttest contrasts were significantly different (*) with $p < .05$. Table 3 shows the effect size, Cohen's d, for each pre- to posttest contrast.

Table 3: Effect sizes (Cohen's d) for each level of test

Group	N	P0–P1	P1– P2	P0– P2
Game	64*	1.48*	-0.30*	1.24*
Control	73*	0.24*	0.57*	0.82*

Cohen's d of 1.48 for the Game group indicates a large increase of 1.5 standard deviations between P0 and P1. However, there was a small decrease for group Game from P1 to P2 of 0.3 standard deviations. Investigation into this drop revealed a significant difference, t (62) = 2.573, $p = .013$, $d = 0.86$, between Computing majors ($M = 3.52$, $SD = 1.69$) and Non-Computing majors ($M = 2.07$, $SD = 1.73$). Figure 7 shows the performance of Computing versus Non-computing majors in the Game group. The difference in P2 is the only significant difference between groups.

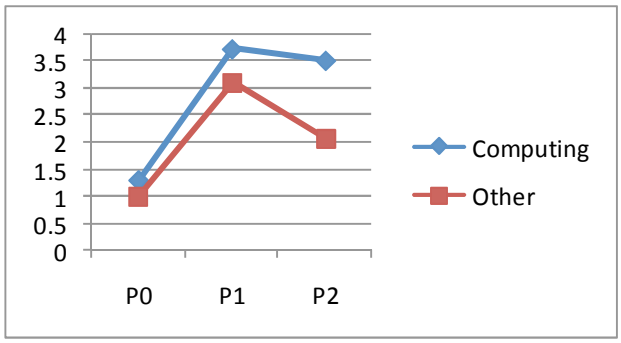

Figure 6: Game group test scores by major

4.2 Test x Group Interaction

The Test x Group interaction was significant F (2, 270) = 30.839, p <.001, Partial η^2 = .19, indicating a difference between the game and programming assignment. The partial η^2 of .19 indicates that 19% of the variance in test scores, beyond what is accounted for by other effects is attributable to the Test x Group interaction. Table 5 contains the contrasts between each level of Group and Test. Group Game (M = 3.63, SD = 1.69) performed significantly higher (t (135) = 5.35, p < .001, d = 0.91), than the Control group (M = 1.95, SD = 1.96) on P1, indicating a higher learning gain from playing the game. Figure 8 shows the means for each group at each level of Test.

Table 4: Pre and Posttest Averages

Group	N	P0	P1*	P2
Game	64	1.243	3.628	3.271
Control	73	1.611	1.945	2.934
Difference p-value		0.092	0.000*	0.301
t-stat		1.695	5.354	1.038

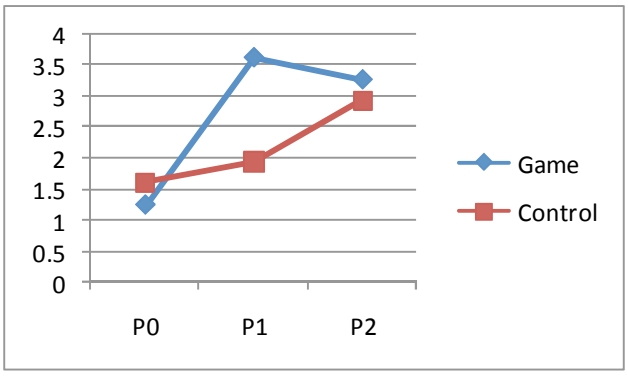

Figure 7: Scores at each level of test for each group

4.3 Main Effect of Group

There was a significant effect due to Group F (1, 135) = 5.028, p = .027, Partial η^2 = .036. This suggests an ordering difference between the groups. The Game group had a higher P2 score (M = 3.27, SD = 1.77) than the Control group (M = 2.93, SD = 2.00), d = .18, though this difference was not significant.

However, submitting the P2 scores to a one-way ANCOVA model, with Group as the between-groups manipulation, and P0 scores entered as a covariate, revealed a significant difference between the groups on mean P2 scores, F (1, 134) = 2.602, p = .025, Partial η^2 = .037, with the Game group adjusted mean (M= 3.43) significantly higher than the Control group mean (M = 2.797). Cohen's d is .39, indicating a medium effect size that without the boost in power from the ANCOVA we would not be able to detect.

4.4 Learning Differences

We submitted the learning differences (P1-P0, P2-P1) between each level of treatment to a 2 x 2 repeated measures ANOVA with Group as the between-subjects factor, and the learning differences as the within-subjects factor. Figure 9 shows the learning differences between tests for each group. The Leaming x Group interaction was significant, F (1, 135) = 60.879, p < .001, η^2 = .311, indicating a difference in learning between the game and programming assignment. With the Game group learning significantly more, t (63) = 9.50, p < .001, after the game assignment (M = 2.38, SD = 1.61) than after the programming assignment (M = -.36, SD = 1.19). The Control group also learned significantly more, t (72) = 2.06, p = .043, after the game assignment (M = 0.99, SD = 1.75) than after the programming assignment (M = 0.33, SD = 1.40).

The main effect of group was also significant F (1, 135) = 6.402, p < .013, η^2 = .045, indicating that order of the assignments affected learning gains. Table 6 shows the differences in learning between the groups.

The P1-P0 scores were significantly higher, t (135) = 7.97, p <.001, d = 1.37, for the Game group (M = 2.38, SD = 1.61) than for the Control group (M = .33, SD = 1.40). The P2-P1 scores were significantly higher, t' (127.51) = 5.31, p <.001, d = 0.59, for the Control group (M = 0.99, SD = 1.75) than for the Game group (M = -.36, SD = 1.19).

A contrast on the overall learning scores (P2-P0) reveals that the Game group (M = 2.03, SD = 1.64) is significantly higher, t (135) = 2.53, p = .013, d = 0.45, than the Control group (M = 1.32, SD = 1.62). This offers further evidence that the game first ordering is preferable for overall learning.

Table 5: Learning Differences between Groups

Group	N	P1-P0*	P2-P1*	P2-P0*
Game	64	2.384	-0.356	2.028
Control	73	0.334	0.989	1.323
Difference p-value		0.000*	0.000*	.013*
t-stat		7.969	-5.185	2.530

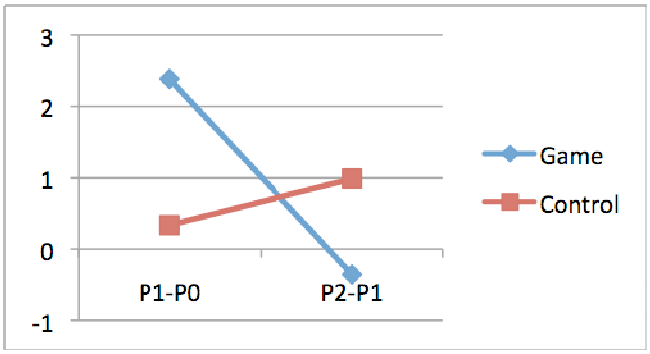

Figure 8: Learning x Group interaction

4.5 Game Logs

Table 7 shows the differences in times for the game and program between the two groups. There was no difference in programming time between the groups; however, the Control group (M = 40:04) finished the game significantly faster, t (135) = 2.22, p = .028, *d-hat* = 0.38, than the Game group (M = 44:34). Each lab was 165 minutes, and the game and program add up to about 74 minutes. Although we intended the pre- and post-tests to take an average of 5-10 minutes each with two students working the tests in parallel, the students took these tests sequentially. Therefore, the students spent about 25 minutes each taking tests and 25 minutes waiting for their partner to take tests.

Table 6: Differences in game and programming time

Group	Game Time	Programming Time
Game (1)	44:34	28:58
Control (2)	40:04	28:31

As shown in Table 8, the game logs did not reveal any major differences in game play. There were 14 missions. Both groups tried each mission about 2 times each and missed about 41% of these on the first try. The Control group was more likely to correct an error on the second try. The Game group had an average of two more errors in the game than the Control group, in the second try, and this partially explains the shorter game times for the Control.

Table 7: Game log data for each group

Group	Tries	Wrong	1st try errors	2nd try errors
Game	31.15	17.15	41%	46%
Control	29.08	15.08	42%	54%

4.6 Qualitative

As shown in Table 9, students overwhelmingly preferred the game to the programming assignment, independent of group.

Table 8: Assignment preference between groups

Group	Game	Program	Both	Neither
Game	48.44%	28.13%	12.50%	10.94%
Control	52.70%	25.68%	10.81%	10.81%

The primary reason students cited for choosing the game was that it was **fun**. The secondary reason was the **interactive feedback** in the game. One student quoted "I like the interactive play and the way you could see what the loops were doing." Some students liked the feeling of advancing through the game, "[The game has] a better feeling of getting something done to advance". Some participants simply enjoyed the imaginative concepts, "It makes me feel like my programming is magical." Many students responded that they liked it "because it was a game."

The primary reason cited for liking the programming assignment over the game was that it was more **applicable** to what they would actually do when programming. The second reason was simply a **joy for programming**. The remaining students either found **both useful**, citing a combination of the above factors, or did not like either assignment.

5. DISCUSSION

The differences in effectiveness of the game over the program dominated the results of this study. After the first intervention, the Game group scored higher than the Control group by almost an entire standard deviation. The Control group did achieve learning gains by writing the program, but they were small in comparison to those in the Game group.

During the second phase of the study, the Game group means dropped by a significant amount. We documented some significant effects of major on the students during the programming portion of the study. Non-computing majors performed 85% of a standard deviation lower on P2 when compared to computing majors, in the Game group. We believe this could be from fatigue from the study, lower motivation to learn programming, or from a failure to immediately transfer game-based learning into actual practice.

The Game group had higher P2 scores when compared to the Control group, but significance was only achieved by factoring out the individual differences from P0. The Game group learned almost half a standard deviation more than the Control group. Therefore, we conclude that the game-first ordering is preferable for overall learning.

However, for non-computing majors in the Game group, a surprising performance drop occurred in scores after writing a program. This result, paired with our results in [8], suggest that some students and particularly non-majors may need time after playing the game before writing a similar program, to allow for some processing of the learning that occurred during the game. More research is needed to see how game-based assignments interact with time and students with different majors and preparation.

There were no differences in programming time between the two groups. Although students are learning theory and gaining skill in analyzing code, they do not seem to immediately transfer this into writing code. This could be explained simply by fatigue; there were a number of tests and tasks to do in a single lab section, and some students had to complete labs from the previous week before starting this lab. Another explanation could be that the added complexity a compiler and IDE can overload students, and particularly non-majors who may not be intrinsically motivated to get programs to work. We do believe that the learning drop we saw for the Game group was permanent; it may be an artifact of the study setup.

6. CONCLUSIONS AND FUTURE WORK

The results of this study agree with our prior research [8], showing that students learn more from Wu's Castle than from a programming assignment. This study shows that even assignments not designed to align with Wu's Castle show similar results. Several important differences exist in these two studies. First, our prior study was conducted asynchronously, with two weeks for students to complete all five steps in the study. Second, our prior study did not allow us to collect data about the time students took in writing their programs. Third, our prior study did not reveal whether any students received any assistance in the course of the study. By conducting the current study in a lab setting, and carefully arranging the study so that each TA taught a Game group and a Control group section, we attempted to control

for non-game or programming interactions. We could also measure the time for each task.

We hypothesized that playing a learning game before writing a related program would improve learning, perceptions, and programming time. We confirmed that students did prefer the game for learning, and learned more using the game, but that programming times were not affected. However, there may be several reasons for this, including fatigue, or a ceiling effect in the amount of time it takes to write a program. We conclude that more study is needed to understand the effects of game-based learning on programming times on long-term learning effects.

The results of this study provide strong evidence for game-based learning in the computing classroom. It also provides evidence that educational games created by undergraduate students can have a meaningful place in these classrooms. We also hope that studies like these will help define game research, with a basis in strong controlled studies.

In our future work, we will continue to build and measure the effects of learning games for computer science, using strong research designs, complete game logs, and qualitative surveys. We believe this trio of metrics can help build a paradigm for robust research into the effectiveness of game-based learning.

7. ACKNOWLEDGMENTS

This work was partially supported by NSF-**0552631** Computing REU Site, and the GAANN grants program at UNC Charlotte.

8. REFERENCES

[1] ACM/IEEE-CS Joint Curriculum Task Force. *Computing Curricula 2001*. Accessed Sep. 8, 2007. http://acm.org/education/curric_vols/cc2001.pdf

[2] Barnes, T., H. Richter, E. Powell, A. Chaffin, A. Godwin. Game2Learn: Building CS1 learning games for retention. *ITiCSE 2007*, Dundee, Scotland, June 25-27, 2007.

[3] Bayliss, J. D. and Strout, S. 2006. Games as a "flavor" of CS1. SIGCSE '06. ACM, New York, NY, 500-504.

[4] Beauboeuf, T & J. Mason. Why the high attrition rate for computer science students: some thoughts and observations. *SIGCSE Bull.* 37, 2 (Jun. 2005), 103-106.

[5] Cooper, S., Dann, W., and Pausch, R. 2000. Alice: a 3-D tool for introductory programming concepts. Mahwah, New Jersey. *Consortium for Computing Sciences in Colleges*, 107-116.

[6] Dale, N. B. Most difficult topics in CS1: results of an online survey of educators. *SIGCSE Bull.* 38, 2 (2006), 49-53.

[7] Eagle, M. & T. Barnes. Wu's Castle: Teaching Arrays and Loops in a Game. *ITiCSE 2008.* Madrid, Spain, July 2008.

[8] Eagle, M. & T. Barnes. Experimental Evaluation of an Educational Game for Improved Learning in Introductory Computing. To appear in *SIGCSE 2009.* Knoxville, TN, 2009.

[9] Gee, J. P. What video games have to teach us about learning and literacy. *Comput. Entertain.* 1, 1 (Oct. 2003), 20.

[10] Hunicke, R., Robison, A., Squire, K., and Steinkuehler, C. Games, learning and literacy. *Sandbox 2006*. ACM Press, New York, NY, 19-19.

[11] Lister, R. 2000. On blooming first year programming, and its blooming assessment. In *Proc. Australasian Conference on Computing Education* (Melbourne, Australia). A. E. Ellis, Ed. *ACSE '00*, vol. 8. ACM, New York, NY, 158-162.

[12] Prensky, M. 2003. Digital game-based learning. Comput. Entertain. 1, 1 (Oct. 2003), 21-21.

[13] Squire, K. (2003). Video games in education. *International Journal of Intelligent Simulations and Gaming*, vol. 2, 49-62.

[14] Wolz, U., Barnes, T., Parberry, I., and Wick, M. 2006. Digital gaming as a vehicle for learning. SIGCSE '06. Houston, Texas. ACM, New York, NY, 394-395.

[15] Zweben, S. 2006-2007 Taulbee Survey. *Computing Research News,* vol. 20, no. 3, May 2008.