Experimental Evaluation of an Educational Game for Improved Learning in Introductory Computing

Michael Eagle
University of North Carolina at Charlotte
Computer Science Department
9201 University City Blvd.
1-704-687-8577
maikuusa@gmail.com

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

ABSTRACT

We are developing games to increase student learning and attitudes in introductory CS courses. *Wu's Castle* is a game where students program changes in loops and arrays in an interactive, visual way. The game provides immediate feedback and helps students visualize code execution in a safe environment. We compared the game to a traditional programming assignment in an introductory CS course. In our study, half of the students were randomly selected to play the learning game first and half to write a program first. Our results show that students who play our learning game first outperform those who write a program before playing the game. Students in the game-first group felt they spent less time on the assignments, and all students preferred the learning game over the program. These results suggest that games like *Wu's Castle* can help prepare students to create deeper, more robust understanding of computing concepts while improving their perceptions of computing homework assignments.

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

Human Factors

1. INTRODUCTION

Computing enrollments have been declining at alarming rates since 2000 [10]. Attrition rates in computing are as high as 30-40%, with most students leaving after taking CS1 or CS2 [2]. Increasingly, educators are using games to motivate students in computing assignments, curricula, and undergraduate research [1][4]. The focus is largely on students developing games. In our two-pronged approach, we engage advanced students in building games to teach introductory computing. Then, introductory students *play* these educational games where learning is in a more enjoyable, relevant context that helps make abstract concepts more concrete. In this paper, we present one student-developed game for teaching arrays and loops, and the effects of this game on learning and attitudes for introductory computing students.

Studies have defined the qualities of games that might be leveraged to improve education (e.g. challenge, curiosity [8]), and investigated the importance of the social context of games in informal learning [6]. However, the study of games has lacked a coherent research paradigm [8], and there is a need for controlled studies to reveal the relative importance of game characteristics for learning. The second author has established the Game2Learn research project, which engages advanced undergraduate computing students in building educational games, and testing these games in introductory computing classrooms. This project is successful in supporting undergraduate retention and recruiting into computing graduate programs [1], while also revealing some important aspects of educational games through user studies [4].

According to a survey of computing educators, loops and arrays are two of the top three programming topics reported to be difficult for novice students [3]. Testing and debugging were two items often mentioned on this survey as well [3]. Compiler errors can be confusing for beginning students, and code that compiles but produces incorrect results can be even more bewildering. Beauboeuf cites both poorly planned labs where teachers debug student code, and lack of practice with meaningful feedback, as causes for attrition in CS [2]. Both of these examples beg for educational systems that assist students in anticipating and finding errors, and providing students individualized feedback.

As an undergraduate student, the first author developed *Wu's Castle* to teach loops and arrays in an interactive, visual way. In *Wu's Castle* the player interactively constructs C++ code to solve in-game problems. The player is able to watch how a program steps through their code and identify logic errors. We hypothesized that 1) playing the game would improve learning over a traditional programming assignment and 2) playing the learning game before writing a program would improve learning. We used a strong "switching replications" experimental design to compare the effects of the learning game with a traditional programming assignment. In the first phase of the experiment, students who played the game achieved higher pre- to post-test learning gains. In the second phase, students in the experimental group wrote a program while those from the control played the game. As hypothesized, students who first played Wu's Castle enjoyed higher learning gains over those who first wrote a program and then played the game, and these gains were marked for more difficult problems. Our findings suggest that playing a learning game *before* writing a program can help students develop deeper, more robust understanding of computing concepts.

SIGCSE'09, March 3–7, 2009, Chattanooga, Tennessee, USA.

2. THE GAME & ASSIGNMENT

Wu's Castle is a 2-dimensional role-playing game developed in RPG Maker XP. The game was created by an undergraduate student using the rapid-prototyping technique as described in [1]. A pilot study of Wu's Castle showed it to have significant effects on learning for CS1 students who played the game as extra credit [4]. In this work, we have expanded the loop walkthrough level and conducted a larger, more robust research study to test the effect of the game on learning both before and after writing a program on loops and arrays. In this section, we briefly describe the game and traditional programming assignments.

In *Wu's Castle*, players interact in two ways: by manipulating arrays by changing loop parameters, and by physically walking the game character through loop execution. After an introduction to the game story and interface, the player walks through a visual representation of a for-, while-, and do-while-loops. The player then manipulates a one-dimensional array by setting the parameters in a for loop (the start, end, and step size). The player repeats this process with a nested loop walkthrough and two-dimensional array manipulation.

Since feedback can positively affect student attitudes and performance in learning games [1], sound and image feedback are immediately given for correct and incorrect responses. Infinite loops cause the game characters to faint; out-of-bounds errors cause an explosion that shakes the screen. After any failure, the level is reset and the player is able to try again. If the player gets the question right in one or two tries, they receive bonus points. Obtaining enough points results in a "level up" animation and sound designed to motivate the player. The player's score is shown in an "experience" meter in the bottom left corner of the screen. The meter and feedback help players understand their progress and learn from their mistakes. *Wu's Castle* records an XML log of all of the player's activities and responses.

Figure 1 shows the cast of characters seen in *Wu's Castle*, including the five snow types the player creates in the arrays, the two characters that guide the player through each mission, and the *machina* character that executes the player's code.

Figure 1: Cast of Wu's Castle

2.1 Array manipulation

In the game, players use for-loops to create arrays of snowmen. An in-game prompt provides the structure of a for-loop and allows the player to change the initial and stopping conditions and the step size. After creating the for-loop, the player selects the body code and watches the code execute. Figure 2 shows the for-loop as the player has set its parameters: for (i=0; i<=19; i=i+1) array[i] = Snowman. The player selected the start, end, and increment values; the *machina*, a game character that executes code instructions, is halfway through creating a row of 20 snowmen.

The missions for both the one and two dimensional array portions of the game progress through changing all elements of an array, to every other element, to a subsection of the array, to every 3rd element of the subsection. The player must complete each mission in order to move on to the next, more challenging mission. This progression was designed to demonstrate typical simple for loops and illustrate how changes in the parameters affect the loops.

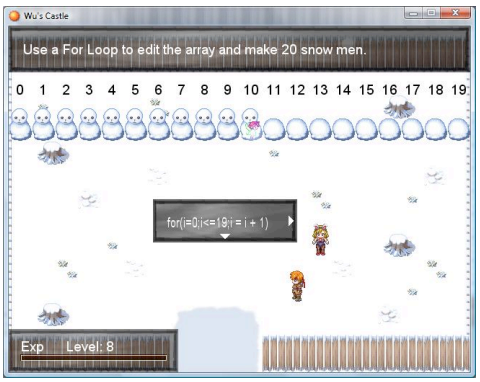

Figure 2: The *machina* executing code, showing half of the array updated during Mission 1

2.2 Loop walkthroughs

In our pilot study, we found an interactive visualization of nested loop execution to be effective for learning [4]. Therefore, we added two more walkthrough examples, one for while loops and one for do-while loops. Figure 3 shows the walkthrough level, where the path is the order of execution, chests represent lines of code, and all variables are displayed on the right. As the player walks by each chest, a line of code executes and all variables are updated. We've numbered Figure 3 to show where each bit of code executes: 1) initialize variables, 2) print "begin", 3) outer loop, 4) print outer loop counter, 5) inner loop, 6) print inner loop counter, 7) print "end". This level was designed to help students visualize and interactively walk through the structure and execution of nested loops. Students begin to realize how inner loops start and how many times they iterate before closing, because their character is not allowed to exit the inner loop until it is complete. This interactive visualization helps students develop a richer understanding of code execution and looping structures.

Figure 3: Nested Loop execution walkthrough

2.3 Traditional programming assignment

The traditional programming assignment was designed so that students completed the same missions, containing 14 tasks in the same order as in the game. Students were provided with scaffold code methods for loops and nested loops. Directions for each

mission were given in commented lines within the provided code. Students were able to complete each array manipulation mission by calling a method with the initial, test, incrementing, and input values as parameters. Students who ignored the scaffold methods and manually programmed each mission were also given credit.

3. METHOD

Our study is a randomized stratified sample with a crossover, or "switching replications" experimental design. In this two-group design, three waves of measurement are taken, one before any treatments, one after each group has tried one treatment, and one after the groups switch and try the other treatment. This strong experimental design can be seen as one pre-test post-test controlled study followed by another replication of the study where the experimental and control groups switch roles. This design removes social threats to validity, allowing all students equal access to the game assignment, but does allow for a testing threat, a learning effect from repeatedly taking the test.

The study was implemented in the second course of a three-course CS1-CS2 sequence when students usually write a program to manipulate arrays using nested loops. Three sections of a class (with the same instructor) were assigned to participate in this study as a homework assignment. All participants took a demographic survey and pretest, and were randomly assigned to one of two groups, evenly distributed based on the demographics and pretest. Each student then performed Task 1, Posttest1, Task 2, Posttest2 and took a post-survey. In the Experimental group, Task 1 is the game, and Task 2 is the programming assignment. The task order is reversed for the Control. Students had a few days to complete each of Task 1-Posttest1 and Task 2-Posttest2.

Table 1: Sample pre- and post-test questions

Q1	Five true/False questions, such as: "In a while loop, the control expression is tested at the end of the loop."
Q2	int Num[7]={0,7,6,2,1,8,5}; What are the values of the expressions below? Num[0], Num[5], Num[4], Num[2]+Num[3]
Q3	What are the values for each expression below? 9 8 7 6 5 Array[3][2] ? 4 3 2 1 0 Array[5][4] ? 1 2 3 4 5 Array[2][1]+Array[0][1] ? 6 7 8 9 0 Array[1][3] ? 5 5 5 5 5

Q4	Given the code below, give the values in the last row of Num.
	int x=7, Num[3][2];
	for (int j = 0; j <= 2 ; j++)
	for (int k = 0; k <= 1 ; k++) {
	Num[j][k] = x;
	x++;
	}

Q5	How many times will Line A be executed?
	int Num[100];
	for (int k = 0; k < 100; k++)
	{
	Num[k] = 1; // Line A
	if (k > 10) break;
	}

Q6	Show on the grid which array elements are checked by this code.
	for(int i = 0; i < 4; i++)
	for (int j = 0; j < 5; i = i + 2)
	array[i][j] = checked;

The pre- and post-tests were designed to be isomorphs, with variable values and question order rearranged between the tests. Sample questions are listed in Table 1. The pre- and post-test measures were designed to test learning at all levels of Bloom's taxonomy using guidelines in [7]. Question 1 is at the knowledge level, 2 and 3 at understanding, while questions 4-6 represent analysis that requires students to predict code outcomes.

The pre- and post-tests and surveys were administered online through (www.surveymonkey.com). The game and programming assignments were assigned online. After each assignment students were asked to email us the game log files and completed programs. Both the game and the programming assignment were constructed to take between 45-60 minutes.

Ninety-two students who completed the pre-test were randomly assigned into 2 groups of 46, after stratifying the sample by gender, race/ethnicity, game habits, and pre-test performance. In each group there were 27 whites, 7-9 blacks, 5 Asians, 2-3 Hispanics, and 0-1 Native Americans. Twenty-six students in the experimental group and 29 in the control completed all three tests, pre-, post1, and post2. Participants identified themselves as casual gamers, hard-core gamers, or neither. Of 13 hard-core gamers in the experimental group and 11 in the control, 8 and 9 respectively completed the study. Of eight women in each group, six women in the experimental and 3 in the control completed the study. Analysis of results was performed using one-tailed t-tests on matched samples and considered significant at the p=0.05 level.

4. RESULTS AND DISCUSSION

Figure 4 shows the results for the pre-test (P0), posttest1 (P1), and posttest2 (P2) over the study for the experimental and control groups. This chart demonstrates a learning effect for both the program and game when administered first, but a larger one for the game. The smaller slopes for the game and program for the control group indicate a dampening effect for the program first.

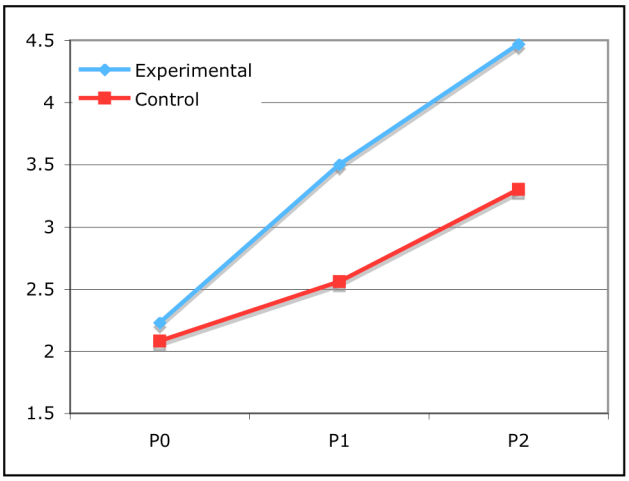

Figure 4: Test results at each phase of the study

Table 2 lists the scores (out of 6) for the Experimental and Control groups for the pre, and posttest 1 and 2 scores. The pre-test scores were not significantly different between the two groups. The difference in the posttest1 and posttest2 scores between the two groups is significant. Table 3 compares the learning differences in the phases of the study, and shows that the experimental group

learned more in phase 1 (between the pre-test P0 and post1 P1) and overall (between P0 and P2). The gains over phase two did not show a significant difference.

Table 2: Pre and posttest averages

	N	P0	P1	P2
Experimental	26	2.23	3.50*	4.47*
Control	29	2.08	2.56	3.30
Difference p-value		0.30	0.001	0.0002
t-stat		0.53	3.12	3.77

Table 3: Learning differences between groups

	N	P1-P0	P2-P1	P2-P0
Experimental	26	1.28*	0.97	2.24*
Control	29	0.48	0.74	1.22
Difference p-value		0.007	0.24	0.0007
t-stat		2.55	0.72	3.36

Table 4 shows the pre- and post-test scores for the Experimental group, who played the game first, broken into students who considered themselves hardcore gamers, men who were not hardcore gamers (Non-HC Men), and women (none of whom rated themselves as hardcore gamers). The pretest was significantly lower for the hardcore gamers in the experimental group ($p=0.04$) when compared to the rest of the students, which corresponds to the lower performance of gamers as found in [9]. However, after playing the game scores among the subgroups were not significantly different. Table 6 lists the same breakdown for the control group, but performance among these groups was not significantly different in any phase of the study.

Table 4: Subgroups for the experimental group

	N	P0	P1	P2
Hardcore	8	1.81*	3.42	4.19
Non-HC	12	2.60	3.59	4.70
Female	6	2.05	3.44	4.38

Table 5: Subgroups for the control Group

	N	P0	P1	P2
Hardcore	9	2.14	2.83	3.47
Non-HC	17	2.07	2.64	3.34
Female	3	1.95	1.35	2.65

Tables 6 and 7 list the game statistics for the experimental and control groups. The hardcore groups tended to have a higher percent correct, spent more time on the game, and had a higher percent done. The play times did tend to be shorter for the control group, who played the game second, but the percent of correct tries was lower for groups in the control.

We ran a correlation analysis to determine the possible effects of game performance and time spent on post-test performance, and we report significant correlations here with $p<0.05$. For both groups, the performance and learning gains for the posttest immediately following the game correlated with the percent complete in the game. There is a correlation between the time spent playing the game and the P1 to P2 learning gains in Phase 2 for the experimental group ($r=0.44$) and for the control ($r=0.46$). This means that with greater time spent on the game, students who played the game first had higher learning gains continuing through the programming phase. Longer times and higher percents complete in the game correlated with higher P2 scores, regardless of when the game was played.

Table 6: Experimental group game statistics

	N	Tries	Correct	Wrong	Time	% Done
Hardcore	8	18.6	13	5.6	33:43	93
Non-HC	12	17.5	12	5.5	33:06	86
Female	6	16	9.2	6.8	29:47	66
TOTAL	26	17.5	11.7	5.8	32:31	84

Table 7: Control group game statistics

	N	Tries	Correct	Wrong	Time	% Done
Hardcore	9	15.4	10.9	4.5	31:30	78
Non-HC	17	17.1	10.2	6.9	30:41	73
Female	3	21	11	10	37:29	79
TOTAL	29	17	10.5	6.5	31:38	75

Table 5 shows the performance of each group on the tests P0, P1, and P2, with the percent of correct experimental group scores in light grey (G1) and control in black (G2). Since the questions are ordered by difficulty level and Bloom's taxonomy level, the graph shows larger performance gaps between the experimental and control groups as questions become more complex.

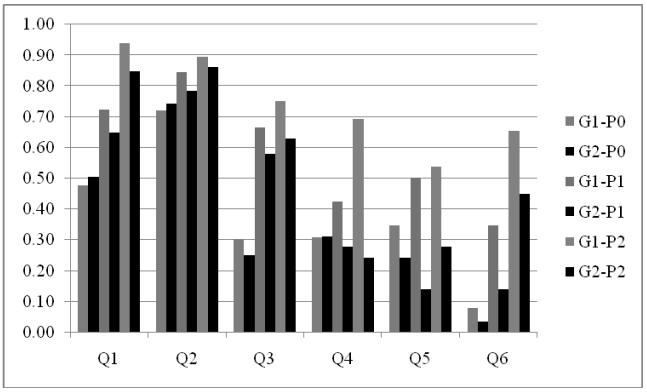

Figure 5: Performance by question

4.1 Survey results

Table 8 shows 10 five-point Likert-scale survey questions asked about each task (program or game) at the end of the study. For all items but 3, 4, and 10, the program was rated lower than the game, with ratings by the control group lower for both the program and the game. On item 10, the experimental group rating for writing more programs was lower than that for the control. For items 3 and 4, both groups rated the program as being more confusing and harder to start at first by both groups.

Table 8: Survey items, asked for each task, game and program

#	Survey Item
1	I enjoyed this task
2	This was an easy task
3	Sometimes I was unsure of what I was supposed to do.
4	At first it was hard, but it was easy after I got the hang of it
5	I liked creating code in this task
6	Overall, this task was helpful in learning computer science
7	This task helped me understand Loops.
8	This task helped me understand Arrays.
9	This task helped me understand Nested Loops.
10	I would like to do more tasks like these

We also asked students to select the amount of time spent on each assignment, in increments of 10 minutes from 20-70 minutes. The average of the reported times for the experimental group were 33 minutes on the program and 29 minutes on the game, while their average actual game time was 32.5 minutes. The average of the reported times for the control group were 40 minutes on the program and 38 minutes on the game, and their actual average game time was 31.5 minutes. Although this measurement is not precise, students felt the game was shorter than it was when they played it first, and felt it was longer than it was when they played it last. All students felt the game was somewhat shorter than the program. When they played the game first, students learned more, and felt that the game took less time than it actually did.

When asked which assignment they preferred, 73% of the experimental and 76% of the control groups prefer the game. The primary reason given for choosing the game over the program was that it was fun. The secondary reason was the interactive feedback in the game. Other reasons included the visual nature of the game, that it helped understanding and illustrated concepts, and several students seemed to appreciate the variety of presentation and forgetting about grades while learning. The positive responses can be understood from one quote: "It was both fun, and very effective. It walked you through the learning process and didn't let you continue until you had mastered each element. This is critical to learning 'the right way'".

On the other hand, students who preferred the traditional assignment (15% of the experimental group and 10% of the control) noted that it was more applicable to the class and like what they would actually do in a program. The remaining students didn't prefer one assignment over another. Of these students, about half found them equally useful and about half found the assignment boring or pointless, while two reported problems with the game when too many incorrect answers were tried. One student who liked both tasks stated: "I feel that both did their purpose very well. The game was helpful to see what loops and arrays do step by step. The traditional assignment was good for seeing how they act in an actual program." From the responses, we concluded that the game was very useful for students who did not already have a clear understanding of loops, arrays, and nested loops. A few more advanced students felt the game was too easy. We could adapt to individuals and allow these students to skip more tedious parts of the game.

5. CONCLUSION AND FUTURE WORK

Our findings show that Wu's Castle is more effective than a traditional programming assignment for learning to solve problems on loops, arrays, and nested for loops. Our results indicate that the game was particularly effective for hard-core gamers in the experimental group in the first phase of the study, bringing gamer and women's scores up to a level with other students after playing the game. This confirms the results we found in our pilot study [4]. Our switching replications experimental design allowed us to confirm our hypothesis that playing a conceptual learning game before writing a program can be helpful in learning. The experimental group who played the game first ended with significantly higher posttest 1 and posttest2 scores and higher learning gains overall. Those students who spent more time on the game and completed more of the game were more likely to perform better overall. Students' perceptions of time spent overall were shorter for students playing the game first. The slope of the learning was also increased in both phases of the study for the experimental group, and the gains were greatest for the most difficult problems. These results suggest that students may be developing deeper, more correct and robust conceptual models for arrays and nested loops when they play a learning game before writing a program, and may spend less time on the pair of assignments overall than when done in reverse.

6. ACKOWLEDGMENTS

This work was partially supported by the CRA Distributed Mentor Project and by NSF-**0552631** Computing REU Site at UNCC.

7. REFERENCES

[1] 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.

[2] 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.

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

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

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

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

[7] 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.

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

[9] Wilson, B. and Shrock, S. Contributing to success in an introductory iomputer science course: A study of 12 factors. *SIGCSE 2001*. ACM Press, New York, NY, p. 184-188.

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