Wu's Castle: Teaching Arrays and Loops in a Game

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

Advisor: 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 teach introductory computer science concepts to increase student motivation and engagement in learning to program. *Wu's Castle* is a two-dimensional role-playing game that teaches loops and arrays in an interactive, visual way. In this game, the player interactively programs magical creatures to create armies of snowmen. The game provides immediate feedback and helps students visualize the execution of their code in a safe environment. We tested the game in a CS1 course, where students could earn extra credit to play *Wu's Castle*. Our results show learning gains for game players, compared both through pre- and post-tests differences and improved performance on relevant final exam questions when compared to students who did not play the game. The results of this study suggest that *Wu's Castle* implements good practices for teaching programming within a game.

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

Keywords

CS1 education, Game2Learn, iteration, arrays, games

1. Problem and Motivation

Enrollments in computing have been declining at alarming rates since 2000, and the proportion of underrepresented groups is also decreasing [1]. Attrition rates in computing are as high as 30-40%, with most students leaving after taking either CS1 or CS2 [2]. Increasingly, researchers and educators are calling for improvements in computing education and have turned to games in order to motivate students through assignments, curricula, and undergraduate research [3-11]. On the other hand, the study of games has lacked a coherent research paradigm [12]. Although studies have defined the qualities of games that might also be leveraged to improve education (such as challenge, fantasy, and curiosity [12]), and researchers are beginning to investigate the importance of the social context and communities built around games in informal learning [13, 14], there is still a need for controlled studies that might reveal the relative importance of game characteristics in motivation and learning. In other words, we are interested in investigating whether simple games that teach computing might demonstrably motivate students to engage in learning outside of the classroom.

1.1 Background and Related work

Dr. Tiffany Barnes has established the Game2Learn research project, which engages advanced undergraduate computing students in building small games that teach individual CS1 concepts, and testing these games in introductory computing classrooms. This project has been successful in supporting undergraduate retention and recruiting into computing graduate programs [15], while also revealing some of the most important aspects of educational games through our user studies [16].

We target our games to novice computing students who are also novice game players. 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 [17]. Testing and debugging were two items often mentioned on this survey as well [17]. 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.

We present *Wu's Castle*, a game developed as a Game2Learn capstone project [16] at the University of North Carolina at Charlotte. *Wu's Castle* was developed 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 tested the game in a CS1 course where students could earn extra credit for playing *Wu's Castle* and taking a pre- and post-test. Questions similar to the pre- and post-test were placed on course's final exam. Students who played the game achieved pre- to post-test learning gains and also outperformed other students in the class on the array section of the final exam. Our results suggest that *Wu's Castle* implements good practices for teaching programming within a game.

2. Uniqueness of the Approach

Wu's Castle was developed in *RPG Maker XP*, a 2-dimensional role-playing game development program that allows developers to quickly build games through drag-and-drop map and event editing and *Ruby* script programming. The game was developed using a rapid-prototyping and testing iteration cycle as described in [15].

Wu's Castle contains two main types of interaction: one for manipulating arrays by changing For Loop parameters, the other for physically walking the game character through the execution of nested loops. In the game, the player is first introduced to

In the underlying code, the student's constructed For Loop is run to modify an array that controls the snowmen drawn at each position. This dynamic implementation allows the player to visualize the actual effects of changing loop parameters. We carefully adjusted the timing of the animation so the *machina* would not travel too quickly; making the visualization more clear.

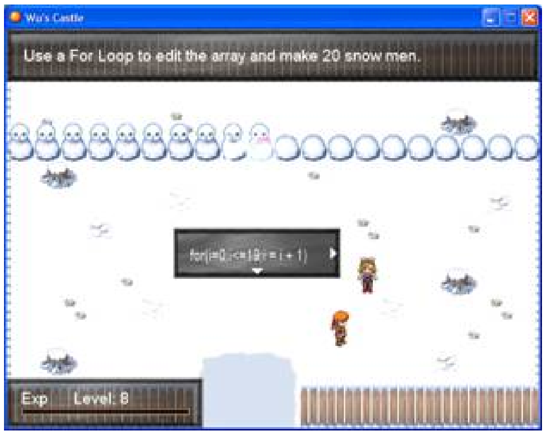
the story and the game interface. In Level 1, the player explores one-dimensional array manipulation using for loops. In Level 2, the player walks his or her character through a nested loop, and the player is asked multiple-choice questions about the loop's code; such as, "Which variable controls the outer loop?" In Level 3, the player uses nested for loops to modify two-dimensional arrays.

2.1 Array manipulation (Levels 1 & 3)

In Levels 1 and 3 the player uses a magical creature called a *machina* to create armies of snowmen and escape a mirror universe. To build the snowmen, the player must manipulate an array of values that describe the type of snowman at each position. The array is represented in the game as a line or grid of snowman characters. In each position, there might be no snowman, a regular snowman, a snowman with arms, a snowman with a hat, or a knocked-over snowman.

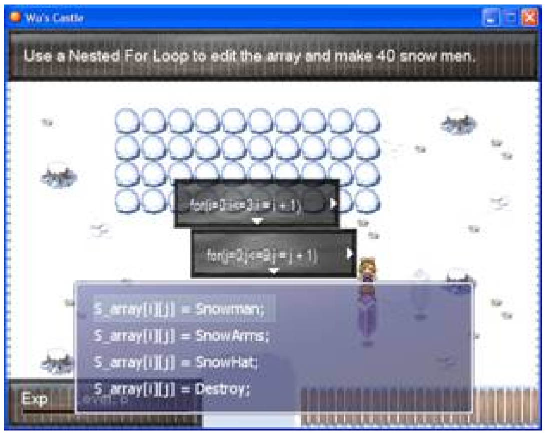
Wu's Castle provides the structure of a For Loop in C++ and allows the player to change the initial condition, stopping condition, and step size. The player then chooses the body of the loop and watches the loop execute. The player's code is a set of instructions telling the *machina* which array positions to visit (through loop parameters) and what snowman to place in each position (through the loop body). As she executes her instructions, the player watches how she changes the array. Her travel through the array visualizes how the parameters of the For Loop control her movement, and what positions in the array are visited. Her animation placing a certain type of snowman at each position demonstrates the body of the For Loop.

Figure 1 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, and the machina is halfway through creating a row of 20 snowmen. Figure 2 shows the player choosing the body of a nested loop to create 40 snowmen.


Use a For Loop to edit the array and make 20 snow men.

```
for(i=0;i<=19;i=i+1)
```

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


Use a Nested For Loop to edit the array and make 40 snow men.

```
for(i=0;i<=3;i=i+1)
for(j=0;j<=9;j=j+1)
S_array[i][j] = Snowman;
S_array[i][j] = SnowArms;
S_array[i][j] = SnowHat;
S_array[i][j] = Destroy;
```

Figure 2: Selecting the body of a nested loop in Level 3

Table 1 shows Level 1 missions. The player must complete each mission in order to move on to the next, more challenging mission. Level 3's missions follow a similar progression through filling a whole 2-dimensional array, certain rows, certain columns, and finally combinations of subsections, rows and columns. This progression was designed to demonstrate typical for loops and illustrate how changes in the parameters affect the loops.

Table 1: Level 1 missions showing loop parameters

Mission	Start	Stop	Step
Edit the entire array (Figure 1)	0	19	1
Edit even array positions	0	19	2
Edit odd positions	1	19	2
Edit a subsection of the array	3	17	1
Edit every 3rd pos. in subsection	5	10	3

2.2 Nested Loop Walk-through (Level 2)

This part of the game offers the player a worked example. The player is shown a typical C++ program that contains a nested loop with a cout statement. The player can view the code at anytime by pressing a button and is sometimes asked questions about the sample code; such as "which variable represents the outer loop?"

As shown in Figure 3, the map contains two circular paths, representing the two loops in the program. The code statements are represented as chests (1-7) on the path. When the player touches a chest, a part of the code will be executed. The values of the loop control variables are constantly displayed on the screen to show how the variables change during each step of the program.

When the player walks around the large (outer) loop, they are directed into the smaller loop where they must circle several times before they are able to escape back to the outer loop.

The student walks through the nested loop execution, starting at the top of the diagram. The player progresses downward and enters the outer loop. The game indicates which line of code is being executed each time the player touches a chest. When the player enters the inner loop the path out is blocked until the player completes the inner loop the correct number of times. The labels in Figure 3 show program statements as described below:

1. Initializes the variables

2. Prints "BEGIN"

The player's score is displayed as an "experience" meter in the bottom left of the screen. The player is awarded with experience points for correct answers and completed missions. Bonus points are awarded when a mission is completed in one or two tries. Incorrect answers can sometimes result in a subtraction of points; the amount varies depending on the severity of the incorrect response. When the bar becomes full the player's rank is increased, and an animation and sound is played. This rank is used to motivate the players to perform well.

Wu's Castle records a time-stamped log for each game session. At the start of a game session, the player is asked to enter a user ID. The log file is placed into the

3. Enters the outer loop
4. Prints the outer loops variable
5. Enters the inner loop
6. Prints the inner loops variable
7. Prints “END”

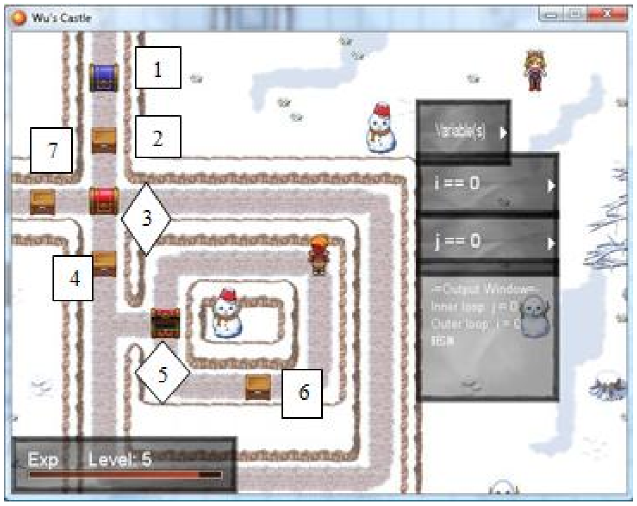


Figure 3: Nested Loop execution walk-through

This level of the game was designed to help students visualize and interactively walk through the structure and execution of nested for 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.

2.3 General Game Features

In our previous studies, we found that prompt, obvious feedback positively affected student attitudes and performance in learning games [16]. Due to the importance of feedback, in *Wu's Castle*, the player receives feedback after each attempt at an answer. When the solution is correct a large green circle is displayed accompanied by a positive sound. If the student’s code fails to compile the *machina* falls unconscious, this represents a syntax error. and the student must try again. An incorrect solution results in a large red X and a negative sound, representing a logic error. If the code goes out of the array’s bounds, the *machina* explodes, representing a program crash. After any failure, the array and *machina* are reset. Should the player complete a mission in one or two tries a bonus animation, and sound are played. The obvious feedback helps novice game players understand their progress.

game directory when the game exits. The log file is lightly encrypted to prevent tampering.

3. ARRAY LEARNING STUDY

Twenty-eight students in a CS1 class were offered extra credit for playing *Wu's Castle*. The students were asked to sign an informed consent form, take a pre-test, play the game, and then take a post-test and qualitative survey. The course’s final exam was given later, including questions listed in Table 2, which were similar to those on the pre- and post-tests.

Table 2: Array questions; the questions on the pre- and post- tests were similar, with some numbers changed.

Q1	Given the following code:
	int Num[20];
	for (int k = 0; k < kMax ; k++)
	   Num[k] = k+1;
	for (k=kMax-1;k >=0 ;k--)
	     cout << Num[k] << endl;
	Which values of kMax will cause an out-of-bounds error
	
	(Circle all that apply)?
	  9, 10, 11, 19, 20, 21, 10000,
	  No values will produce an out-of-bounds error,
	  There is no such thing as an out-of-bounds error

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	Given the following code, draw the array Num with its values.
	  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++;
	     }

Q4	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;
	  }

Q5	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

The pre- and post-tests and qualitative survey were administered through an online surveying tool (www.surveymonkey.com). The game was made available online, and students were able to download and play it at home. The students first took the pre-test and then played the game. After playing the game, the students were asked to email the log files to the first author, and take the online post-test and survey. There were no set requirements on how long to play the game, but we expected the game to take about 20-30 minutes.

4. Results and Contributations

Since the study was conducted as extra credit at the end of the semester, the level of participation varied across the class. Of 27 students in the class, 11 did not play Wu’s Castle, or take any pre- or post-tests; we call this group Control-NoPretest

Table 4: Pretest scores for groups Game (N=9) and cP (N=7)

	Q1	Q2	Q3	Q4	Q5	Tot.	%
Game	0.4	0.475	0.1	0.3	0.375	1.65	33
cP	0.64	0.17	0.50	0.33	0.21	1.85	37

Table 5: Average of (final-pretest) score differences

	Q1	Q2	Q3	Q4	Q5	Total	%
Game	0.41	0.38	0.15	0.10	0.10	1.14	22.8
cP	0.08	0.17	-0.33	-0.17	0.00	-0.25	-5

(cNP). Seven students took the Array pre-test but did not play Wu's Castle; this group is Control-Pretest (cP). Nine students completed the pre-test, played Wu's Castle and turned in a log file, and completed the post-survey; this is group Game. Table 3—7 lists the pretest, posttest, and final exam scores for each group. Each question was worth one point. Questions with multiple part-answers allowed students to earn partial credit for each correct part.

4.1 Pretest to Posttest Learning Gains

Table 3 shows the average scores on each question and overall for group Game on the pre- and post-tests and corresponding final exam questions. For the Game group, the average scores on the posttest were statistically significantly higher than the pretest (2.70 to 1.65) (p=.003). Comparing the difference between an individual student's posttest and pretest score reveals an average 17% grade improvement. Overall, 8 of the students in group Game improved their scores, while 1 student showed a slight decrease in scores between the pre- and post-tests. Students (N=4) who made it to the second level of the game performed significantly better on question 5 (p=.03) on the posttest. The variance decreased from 1.73 to 0.49 between the pre and posttest. The average posttest scores compared with the similar questions on the final exam were slightly higher, indicating that the students retained knowledge.

Table 3: Results for group Game (N=9)

	Q1	Q2	Q3	Q4	Q5	Total	%
Pretest	0.40	0.48	0.10	0.30	0.38	1.65	33
Posttest	0.51	0.75	0.67	0.33	0.44	2.70	54
Final	0.81	0.85	0.25	0.40	0.48	2.79	55.8

4.2 Pre-test to Final Exam Comparison

Table 4 lists the pretest performance of the groups Game and cP. Overall, the average pretest scores for these two groups were not statistically significant (p=.35). For each student, we computed the difference between the final exam and pre-test score for each question, and the average of these differences over all students is shown in Table 5. We note that while group Game performed significantly better on the final exam array questions than on the pre-test, the difference in final and pre-test scores was not significant for group cP (p= .34). This shows that exposure to the pre-test questions did not have a significant effect on performance on the corresponding questions on the final exam for group cP.

Table 6 shows the performance on the final exam for the questions corresponding to the pre- and post-tests for each group. There was no significant difference between the control groups on the final exam questions (p=.37). Together with the pre-test to final exam comparison for group cP, these results provide evidence that there was no significant effect from multiple exposures to similar questions.

Table 6: Final exam comparison between control groups

	Q1	Q2	Q3	Q4	Q5	Tot.	%
cP	0.72	0.33	0.17	0.17	0.21	1.60	32
cNP	0.47	0.36	0.18	0.09	0.27	1.38	27.6
cP+cNP	0.59	0.35	0.17	0.13	0.24	1.49	29.8
Game	0.81	0.85	0.25	0.40	0.48	2.79	55.8

4.3 Final Exam Performance

Table 7 shows the percent correct for the array portion, the non-array portion, and the overall final exam for all three groups. The array portion of the exam consisted of the 5 pre- and posttest questions plus 2 additional questions. Although all scores were higher for group Game, none of the score differences are significantly different, except for those on the array portion. Group cP's scores were also higher than those for group cNP, though this difference is not significant.

Performance on the array section of the final exam was almost equal between cP and cNP; however, cP scored about 5 points higher on the remaining portion of the final. The self-selected students (group Game and cP) performed 11% higher on the non-array section of the final exam; while there was no statically significant difference found, the self-selected students may have been slightly better performers.

Table 7: Final exam performance by section of the final exam

	N	Array Portion	Non-Array Portion	Total
Game	**9**	**54.63**	78.35	77.16
cP	7	37.43	75.98	71.37
cNP	11	36.71	70.42	66.93
Control	18	37.05	73.20	69.15

When comparing Game and Control (cP+cNP), the average score for the **array portion** of the final exam was 51% higher for students who played *Wu's Castle* (p=.01), suggesting that Wu's Castle improved student learning about arrays.

4.4 Qualitative Survey

The qualitative survey consisted of eleven Likert-scale items as listed in Table 8. The scale was a 5-point scale from Strongly Disagree to Strongly Agree. There were also three open response questions that asked the student what part of the game they liked the best, and for suggestions or comments. Table 6 lists the number of students who selected "Agree" or "Strongly Agree" with the listed survey items.

The majority of group Game enjoyed playing *Wu's Castle* and would like to play more games like it. The students also felt as though the game was serious enough for homework in a CS course. Students generally found the game to start out at a doable level, but did not find it too easy. Most did not guess answers in the game; but were motivated to get the correct answers.

Table 8: Percent agreement with survey items (9 participants)

Survey Item	**% agree**
I enjoyed playing Wu's Castle	67
This was an easy game to play	44
Sometimes I was unsure of what I was supposed to do in the game	33

5. CONCLUSION AND FUTURE WORK

Our results showed that students who played *Wu's Castle* achieved statistically significant learning gains when compared to students who did not play the game. Although the sample size was small, the differences in scores were large. These results suggest that Wu's Castle implements good practices for teaching programming within a game; these practices include frequent and obvious feedback and interactive visualization of code. On average, students played the game for about 40 minutes, demonstrating that students appreciate many chances to try the same problems. The results of this study should encourage instructors to consider implementing these technologies in classrooms. Future games can be developed with principles similar to *Wu's Castle* for other concepts in computer science; and give players a place to practice programming and easily identify mistakes in a safe environment. We plan to continue creating Game2Learn games and running studies to track learning and retention rates in introductory computer science courses.

Wu's Castle was Game2Learn's first attempt at online delivery. Some of the players reported difficulty installing the game. Emailing the log files is troublesome; and we only know if students are playing when they email us a log file. Future work should include an easier game installer, more concise install instructions, and a way for the

At first the game was hard, but it was easy after I got the hang of it	11
I liked creating code in the game	67
Overall, this game was helpful in learning computer science	78
Overall, the game had a good balance between "play" and "quest" time	56
I would like to play more quests like these	67
I was motivated to try hard to answer questions correctly in the game	78
Sometimes I missed questions on purpose to see what would happen	11
I guessed at the answers for most of the questions	11

When asked what students liked best in the game, several students said they liked the simple interface. The most common response was that after you made a mistake, you were given the chance to try again with one player saying "It gave me a chance to come up with the answer after I kept getting it wrong and even after I had gotten it right...I could still practice more."

In the recommendations for improvement, almost all of the students desired a better hint system. The game allows players to re-try a mission as many times as desired, but offers little in the way of helping the player, should they become stuck.

logs to be automatically sent. Parsing the games log files for data was time consuming and many times had to be performed manually. A XML based log system has since been developed for future versions of the game. Future work should also include more specialized feedback when the players repeatedly fail at the same task.

6. References

[1] Zweben, S. 2003-2004 Taulbee Survey. *Computing Research Association Taulbee Survey*, May 2005.

[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] Bayliss, J. The Effects of Games in CS1-3, *Microsoft Academic Days Conference on Game Development in Computer Science Education*, Feb. 2007, 59-63.

[4] Bayliss, J. & S. Strout. Games as a "flavor" of CS1. In *SIGCSE2006*. ACM Press, New York, NY, 500-504.

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

[6] Garris, Ahlers, & Driskell. Games, motivation, and learning: a research and practice model. *Simulation & Gaming*, Vol. 33, No. 4, 2002, 441-467.

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

[8] Gumhold, M. & Weber, M. Motivating CS students with game programming. *Proc. Intl. Conf. on New Educational Environments (ICNEE),* Neuchatel, Switzerland, Sep. 2004.

[9] Parberry, I., Roden, T., & Kazemzadeh, M. Experience with an industry-driven capstone course on game programming: extended abstract. *SIGCSE 2005*: p91-95.

[10] Prensky, M. *Digital Game-Based Learning*, New York, McGraw-Hill, 2001.

[11] Wolz, U., T. Barnes, I. Parberry, and M. Wick. Digital gaming as a vehicle for learning. *SIGCSE 2006:* p. 394-395.

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

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

[14] Steinkuehler, C. Learning in massively multiplayer online games. In *Proc. Intl. Conf. Learning Sciences*, Santa Monica, CA, June 22 - 26, 2004, p. 521-528.

[15] Barnes, T. Powell, E. Chaffin, A. Godwin, A. Game2Learn: Building CS1 Learning Games for Retention. *ITiCSE'07*. ACM Press, New York, NY, 500-504.

[16] Barnes, T., E. Powell, A. Chaffin, H. Lipford. Game2Learn: Improving the engagement and motivation of CS1 students. To appear in *ACM GDCSE'08*.

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