Incremental Game Development in an Introductory Programming Course

Mark A. Holliday
Department of Mathematics and Computer Science
Western Carolina University
Cullowhee, NC 28723
holliday@wcu.edu

Abstract-

The enthusiasm students have for playing computer games can be used in an introductory programming course to increase the enthusiasm and attention that students have for developing problem solving and programming skills. This paper reports on a successful experience using incremental development of two computer games, Master Mind and Minesweeper, in the programming assignments of such a course. By the end of one semester of programming instruction the students are able to design and implement programs of functionality equivalent to well-known commercial games. The incremental game development technique can be used either in a procedure-oriented course or in an object-oriented course; assignment sequences for both type courses are presented.

1 Introduction.

Though there is considerable debate about the proper content and organization of the early courses in the computer science major, it is clear that developing problem solving and programming skills is one of the essential goals of these courses. One of the keys to achieving this goal is the enthusiastic interest of the student in the programming assignments. Students have a well-known enthusiasm for playing computer games. We decided to investigate if we could channel this enthusiasm towards the goal of improving problem solving and programming skills.

We felt that to be successful this approach needed two features. First, the end result must be a game that is comparable in functionality to games that the students normally play. Producing a game at that level of functionality clearly makes using it during the debugging stage more fun. Just as importantly, however, being able to program a game at that level of functionality gives the students a very positive message about the power of programming and of their abilities as programmers. Second, in a first programming course few students would be able to develop a program for the type of game we had in mind without any preparation. Consequently, we decided that incrementally developing the game with achievable and interesting milestones was essential.

The above rationale for our approach is consistent with several themes in the computer science education literature on CS1, the first course for computer science majors. One theme is the importance of developing substantial programs in CS1. Substantial programs more accurately reflect realistic programming environments than having a large number of small programs each testing a separate language feature [6]. More substantial programs also reinforce the importance of software engineering and lead naturally into a more formal treatment of software engineering in later courses [10]. The development of the substantial programs in an incremental fashion also reflects good software engineering practice [7]. Good projects should also have visual impact [9] which is consistent with a game-based approach since many games (such as our end result game) have a visual component.

During the Spring 1994 semester we tried using this approach in one of the sections of our CS1 course. The end result game we chose is *Minesweeper*. Minesweeper is one of the two games supplied with every new copy of Microsoft Windows 3.1 system [4]. Consequently, many of the students in the class were already familiar with and have played Minesweeper. We felt that Minesweeper was a good choice since it is a popular game as well as nontrivial.

For the incremental development resulting in the complete Minesweeper game we chose five steps. The first three steps involved implementing a simpler game called *Master Mind.* The fourth step involved implementing a one-dimensional version of Minesweeper. The complete two-dimensional Minesweeper was the fifth step.

The experiment was highly successful based both on student completions and student comments. In this paper we describe the experiment and some extensions. Section Two describes the games Mastermind and Minesweeper as well as the five steps we used in the Spring 1994 course. The course content, student composition, and results of the experiment are presented in Section Three. A major issue currently is whether the teaching of imperative programming languages should use the object-oriented paradigm or the procedural paradigm [1]. During the Spring 1994 course we taught the C++ programming language following the procedural paradigm (as a "better C"). The key approach of a commercial quality, end-result game developed in an incremental sequence is independent of whether the object-oriented paradigm or the procedural paradigm is used. Section Four describes our recent preliminary experiences on keeping the two games but modifying the incremental steps to follow the object-oriented paradigm. We conclude in Section Five.

2 Games and Assignments

2.1 The Games

Master Mind is a popular peg game where one person (the computer) secretly and randomly selects four pegs from a pile of six different colors (selection is with replacement so colors can be repeated). The goal of the player is to guess the colors and positions of the four pegs selected in as few moves as possible. Upon each guess the computer gives clues as to which pegs are correct according to the following rules:

- For each input peg that has a selected color but is in the wrong place, the computer prints an asterisk "*".
- For each input peg that has a selected color and is in the correct place, the computer prints a dollar sign "$".
- For each input peg that is of a color that was not selected for any position, nothing is printed.

To prevent there being a one-to-one correspondence between clues and positions, all the asterisks are first printed and then all the dollar signs. Thus the game's

```
Welcome to Mastermind.
Please enter guess.
PGRO
1 PGRO **
Please enter guess.
OGPP
2 OGPP *$$
Please enter guess.
GPPP
3 GPPP $$$$
Congratulations, you have won the game!
Would you like to play again (y/n)?
n
Thanks for playing Master Mind.
```

Figure 1: A sample session playing any of the versions of Master Mind. Input is shown in boldface.

output consists of successive single lines with each line containing, in order,

- an integer counting how many times the player has guessed in the current game,
- a string of four characters echoing the preceding guess,
- a string of clues consisting of up to four askerisks and dollar signs.

Figure 1 displays a sample session of Master Mind using one of the assignment implementations. The implementation allows the game to be played multiple times during a single program execution. We first learned of the Master Mind from Professor Henry Greenside of Duke University who has used a single assignment version of implementing the game in an introductory Pascal course [3].

Minesweeper is a board game using an 8 x 8 grid that contains mines on ten of the squares. The player attempts to identify the location of each mine. The goal is to uncover all the squares before making an incorrect guess. Initially all the squares are covered. The player selects a square and guesses whether the square does or does not contain a mine. If the guess is incorrect, the game ends. If the guess is correct, the square is uncovered. The uncovered square shows a mine (denoted by the letter 'M') if the correct guess was that the square contains a mine. The uncovered square shows an integer if the correct guess was that the square does not contain a mine. The integer is the number of adjacent squares containing mines; thus, this number ranges from zero to six since a square can have up to six adjacent squares. Figures 2 and 3 display a (unusually brief) sample session

```
Welcome to Minesweeper.
Do you want to see the mines (y/n)?
y
The solution mine grid is:
0 0 1 1 1 0 1 M
0 0 1 M 1 0 1 1
0 0 1 1 1 0 0 0
1 1 0 0 1 1 0 0
M 2 1 1 1 M 2 1
2 3 M 2 1 0 1 M
2 M 4 M 1 0 1 1
2 M 3 1 1 0 0 0
Enter row of guess
(value must be 0 to 7).
5
Enter column of guess
(value must be 0 to 7).
1
Do you think there is a mine at
this square (y/n)?
n
Correct guess!
The current grid is:
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- 3 - - - - - -
- - - - - - - -
- - - - - - - -
```

Figure 2: The first part of a sample, unusually brief, session playing the two-dimensional version of Minesweeper from the fifth assignment. Input is shown in boldface.

```
Enter row of guess
(value must be 0 to 7).
5
Enter column of guess
(value must be 0 to 7).
3
Do you think there is a mine at
this square (y/n)?
n
You just blew yourself up!
Would you like to play again (y/n)?
n
Thanks for playing Minesweeper.  End of game.
```

Figure 3: The continuation of the sample session playing the two-dimensional version of Minesweeper from the fifth assignment. Input is shown in boldface.

playing an implementation of Minesweeper from the last assignment.

2.2 The Assignments

With Master Mind and Minesweeper we felt that we had identified two games that the students would enjoy playing enough so as to help motivate an enthusiastic attempt at the assignments. The next step was to develop a sequence of assignments of increasing language complexity that followed a procedural approach to learning the "better C" fraction of the C++ programming language.

Since an implementation of Minesweeper requires facility with two-dimensional arrays the Master Mind assignments are first. All three of the Master Mind assignments require a complete working solution. However, the second and third assignments require the use of additional language features which encourage more modular code. The Minesweeper implementation is first done for a one-dimensional array version in order to isolate the complexity of dealing with two-dimensional arrays. More specific descriptions of each assignment follow. The handouts and the source code for the assignment solutions are available by electronic mail from the author.

- *Assignment 1.* Implement Master Mind assuming that selection (the *if* statement) and iteration (the *while* statement) have been introduced, but not functions.

- *Assignment 2.* Modify the first Master Mind implementation to be modular which assumes that functions, local versus global variables, and parameter passing have been introduced in class.

- *Assignment 3.* Modify the second Master Mind implementation by storing variables (such as answer peg and guess peg colors) in arrays and indexing over the arrays when appropriate.

- *Assignment 4.* Implement a one-dimensional version of Minesweeper with one row of eight squares and three mines.

- *Assignment 5.* Implement the two-dimensional version of Minesweeper.

Our organization of the three Master Mind assignments might be controversial. Given that we were covering only the "better C" portion of C++ an issue is the order of presentation of functions versus selection and iteration. A widely-held view is that since encouraging modular programming is important, functions should be introduced first. However, in agreement with Pattis [8], we are not convinced that the importance of modular programming implies that introducing functions early is pedagogically best.

In fact, we feel that introducing selection and iteration first is preferable for three reasons. First, selection and iteration, being straightforward flow-of-control concepts, are easier for students to understand early in the course. The introduction of functions forces coverage of scope rules, local versus global variables, and parameter passing which are more complex concepts. Second, is our conjecture that the most effective means of reinforcing the importance of modular programming is by having the student redo a solution that is not modular into a solution that is modular. The importance of a tool, functions, for modular programming is much clearer when it has to be done without. Third, is that it is difficult to assign interesting programming projects that do not require selection and iteration. Introducting selection and iteration early allow the students to start as early in the semester as possible on nontrivial assignments such as Master Mind.

Consequently, the first Master Mind assignment only requires knowledge of basic control flow statements. The second Master Mind assignment uses modular use of multiple functions, local variables, and parameter passing to improve the solution. The step from the second to the third Master Mind assignment is less controversial since arrays are typically introduced after both functions and selection and iteration.

If Master Mind always has the same values for the correct peg colors and Minesweeper always has the same values for the mine locations, the games are clearly less interesting. Our solution is to derive the initial values from a pseudo-random number generator. To prevent always using the same pseudo-random sequence, each time the program runs the seed of the pseudo-random generator is reset using an arbitrary but varying value. The value we use for resetting the seed is a library function call that returns the current time in seconds. The code for including the necessary library and the function calls for setting the seed and generating a random number are included in the programming assignment handouts.

Major	Enrollment
Undecided	7
Mathematics Education	6
Electronics Engr. Technology	4
Computer Science	4
Other	2
Mathematics	1

Table 1: Distribution by declared major of the students in the Spring 1994 section.

3 Course Context and Experience

During the Spring 1994 semester we experimented with the incremental game assignment sequence in one of the sections of the programming and problem solving course. This course attracts a wide range of students with differing degrees of previous exposure to computers and with different goals. There were 24 students in the section as the assignment sequence began divided by major as shown in Table 1. Students planning on teaching mathematics at the high school level were the largest declared major followed by students in the Electronics Engineering Technology major and the Computer Science major. Few of the students had any programming experience.

Table 2 shows for each assignment the number of students who submitted a solution and the mean score of the submissions. Probably at the start of the course, few of the students thought they would be able to implement versions of programs such as Minesweeper. However, as Table 2 demonstrates, 74% of the students were able to complete the two-dimensional version of Minesweeper. Table 2 raises some other interesting points. On individual assignments performance was bimodal. Submissions tended to work and follow good programming style as indicated by the high mean scores. The remaining students tended to "give up" and not submit partially working solutions. The incremental approach of adding language complexity clearly is justified in the cases of both Master Mind and Minesweeper. Almost every student was able to complete the initial versions, while there was a significant decrease in completions as new language features (functions and two-dimensional arrays, respectively) were added. Clearly, many fewer students would have completed a single monolithic assignment.

The quantitative results are encouraging both for the idea of having the end result be an interesting, challenging computer game and for the idea of incrementally developing the game through a sequence of versions with increasing language feature complexity. The anec-

	# Tried	Mean Score
Assg1	23	98%
Assg2	16	96%
Assg3	15	89%
Assg4	21	94%
Assg5	17	96%

Table 2: For each assignment the number of students who submitted a solution and the mean score of the submissions.

todal results are perhaps even more encouraging. For example, one of the students in the class is the mother of a teenage child and the child is an avid player of Minesweeper. When the mother completed her own version of Minesweeper, the child's respect for his mother apparently was raised to an entirely different level. That the students play the games as part of debugging appeared to be a major attraction. Some of these programs may be the most thoroughly debugged programs I have ever assigned.

4 An Object-Oriented Version

The Spring 1994 experience with incremental computer game development was in the context of a procedural approach to the "better C" part of C++. During the Fall 1994 semester we taught CS1 using the C++ programming language, but in an object-based manner. "An object-based manner" means that we introduce classes (and consequently, functions) as early as possible, followed by control structures and then compound data (that is, one-dimensional arrays and then two-dimensional arrays). This follows the sequence of several C++ texts including the text we are currently using [2, 5]. The question naturally arises of how effective incremental computer game development is as a motivational and pedagogical technique in an object-based programming context.

There are two general approaches. One is to maintain the same assignment sequence but to delay the first Master Mind assignment until control structures are covered. The second approach is to make a major revision of the assignment sequence so that a new Master Mind assignment can be assigned early in the course that uses objects but not control structures. During the Fall 1994 semester we followed the first approach.

Attempting to keep the programming assignment sequence the same as during the Spring 1994 semester significantly delayed the first Master Mind assignment. One cause as discussed in Section 2 is that delaying control structures until after functions moves back considerably when the first Master Mind assignment can be assigned. A second cause is that Fall 1994 was the first semester with a new textbook and the object-based approach. It is expected that the next time teaching this course, the first Master Mind assignment can be assigned significantly earlier.

We are also considering the second and more radical approach of redesigning the game assignment sequence to incorporate objects. We now outline some of the issues involved in this second approach. Master Mind still precedes Minesweeper logically since Minesweeper requires arrays. Since implementing Master Mind (as does Minesweepr) fundamentally involves control structures, the key idea is to have the first Master Mind assignment use the Master Mind game object instead of attempt to implement that object.

Implementing the Master Mind game as a C++ class can be done in several ways with interfaces of varying complexity. The least informative is to just have a constructor that contains a loop for each game played and within that loop prompt for and processes guesses. More informative is to have member functions for initializing a new game and for processing each guess. Using this more informative class interface, the first Master Mind assignment becomes a driver program that declares a Master Mind object and then does a sequence of member function calls to initialize a game and then process a fixed number (to avoid looping) of guesses. For each guess the program would read input from the keyboard, pass the guess to the member function, and then display to the screen the result from the member function. By the second Master Mind assignment control structures would have been covered so looping can be introduced in the driver program and the member function implementation can be assigned.

The Minesweeper C++ class and driver program can be organized in a similar manner, but the assignment that just uses the C++ class is not necessary since control structures will have been covered. The sequence of assignments thus becomes as follows.

- *Assignment 1.* Demonstrate understanding of the use of classes by writing a driver program that uses the provided Master Mind class to process a fixed number of guesses in a single game. object module form.

- *Assignment 2.* Use knowledge of control structures for selection and repetition to modify the Master Mind driver program to loop both for guesses within a game and to play multiple games. Also implement the Master Mind class member functions. The implementation need not use arrays.

- *Assignment 3.* Reimplement the Master Mind class member functions using arrays where appropriate.
- *Assignment 4.* Implement a one-dimensional version of Minesweeper as a C++ class and develop a driver program.
- *Assignment 5.* Implement the two-dimensional version of Minesweeper as a C++ class and develop a driver program.

5 Conclusions

For better or worse, many students love to play computer games. As instructors we can turn this attraction to our advantage by using it to motivate our assignments in a problem solving and programming course. To be most effective we think that two traits are necessary. First, the final game the students develop should be at a level of sophistication comparable to commercial games. Having such a target maximizes the students' enthusiasm and respect for what programming can accomplish based upon their own accomplishments. Second, the game should be developed incrementally through a series of assignments. Incremental development allows each assignment to be manageable and allows the assignments to start early in the course.

In this paper we report on our experiences investigating this technique. We used a five assignment sequence developing implementations of the Master Mind and Minesweeper games during the Spring 1994 semester. The experience was encouraging both as measured quantitatively and anecdotally. The incremental game development technique can be used either in a procedure-oriented course or in an object-oriented course. We have outlined assignment sequences for both types of courses.

References

[1] R. Decker and S. Hirshfield, "Top Down Teaching: Object-Oriented Programming in CS 1", *Proc. Twenty-Fourth SIGCSE Tech. Symp. on Comp. Sci. Educ.*, SIGCSE Bulletin, vol. 25, no. 1, pp. 270-273, March 1993.

[2] R. Decker and S. Hirshfield, *The Object Concept*, PWS Publishing, Boston, MA, 1994.

[3] H. Greenside, personal communcation, 1993.

[4] K. Jamsa, *The Concise Guide to Microsoft Windows Operating System Version 3.1*, pp. 156-159, Microsoft Press, 1992, Redmond, WA.

[5] R. Mercer, *Computing Fundamentals with C++*, Franklin, Beedle and Associates, Wilsonville, OR, 1994.

[6] S.R. Oliver and J. Dalbey, "A Software Development Process Laboratory for CS1 and CS2", *Proc. Twenty-Fifth SIGCSE Tech. Symp. on Comp. Sci. Educ.*, SIGCSE Bulletin, vol. 26, no. 1, pp. 169-173, March 1994.

[7] R.E. Pattis, "A Philosophy and Example of CS-1 Programming Projects", *Proc. Twenty-First SIGCSE Tech. Symp. on Comp. Sci. Educ.*, SIGCSE Bulletin, vol. 22, no. 1, pp. 34-39, February 1990.

[8] R.E. Pattis, "The "Procedures Early" Approach in CS 1: A Heresy", *Proc. Twenty-Fourth SIGCSE Tech. Symp. on Comp. Sci. Educ.*, SIGCSE Bulletin, vol. 25, no. 1, pp. 122-126, March 1993.

[9] J. Robergé, "Creating Programming Projects with Visual Impact", *Proc. Twenty-Third SIGCSE Tech. Symp. on Comp. Sci. Educ.*, SIGCSE Bulletin, vol. 24, no. 1, pp. 230-234, March 1992.

[10] J. Robergé and C. Suriano, "Using Laboratories to Teach Software Engineering Principles in the Introductory Computer Science Curriculum", *Proc. Twenty-Fifth SIGCSE Tech. Symp. on Comp. Sci. Educ.*, SIGCSE Bulletin, vol. 26, no. 1, pp. 169-173, March 1994.