**CSCE 2014 – Programming Project 3**

**Midpoint Due Date – Mar 1, 2017 at 11:59pm**

**Final Due Date – Mar 8, 2017 at 11:59pm**

**1. Problem Statement:**

The goal of this programming project is to give students experience with two-dimensional arrays, classes, and recursive functions. Your first task will be to create a “GameBoard” class that stores information about a game board that might be used in a simple video game. This class will have a variety of methods that allow users to store, retrieve and display “game pieces” on the game board, including a recursive “flood\_fill” method. Detailed instructions for the GameBoard class are given below.

Your GameBoard class should store the following information:

* **MAX\_ROWS** – the maximum number of rows in the game board. This constant should be defined at compile time in the GameBoard.h file.
* **MAX\_COLS** – the maximum number of columns in the game board. This constant should be defined at compile time in the GameBoard.h file.
* **Rows** – the number of rows being used in the game board. This variable should be set in the constructor function, or the “read\_board” method.
* **Cols** – the number of currently being used in the game board. This variable should be set in the constructor function, or the “read\_board” method.
* **Board** – a fixed size two-dimensional array of ASCII characters that represents the contents of he game board. Each array location will contain an ASCII character that represents something on the game board. For example, ‘x’ might represent a wall, ‘.’ might represent an empty space, and ‘o’ might represent a creature of some type.
* **Empty** – the ASCII character that is used to mark an empty location in the game board. When the Board array is initialized, all array locations should be set to this Empty character.

Your GameBoard class should implement the following methods:

* **Constructor** – initialize the “GameBoard” object to an empty state. Set all of the private variables of the object to zero or blank characters.
* **Destructor** – initialize the “GameBoard” object to an empty state. Set all of the private variables of the object to zero or blank characters.
* **get\_rows, get\_cols, get\_empty** – methods that return the current value of these private variables.
* **get\_board** – method to get the value of the board at location (r,c). You should check that (r,c) is a valid location before returning a value.
* **set\_board** – method to set the value of the board at location (r,c). You should check that (r,c) is a valid location before saving a value.
* **initialize\_board** – change the size of the board to the specified number of Rows and Cols, save a new value for the Empty character, and initialize the board to contain all Empty characters.
* **print\_board** – print the characters in the Board array on the screen for the user to look at. You may want to print spaces between the board characters and include an outline for the board to make it look nice.
* **read\_board** – open a specified ASCII file, and read the information in that file to initialize a GameBoard object. The first line in the file will contain the values for Rows, Cols, and Empty. There will then be Rows \* Cols characters separated by spaces that represent the contents of the Board array. You should read these values in “row major order”.
* **write\_board** – open a specified ASCII file, and write the information in that file to represent a GameBoard object. The first line in the file will contain the values for Rows, Cols, and Empty. There will then be Rows \* Cols characters separated by spaces that represent the contents of the Board array. You should write these values in “row major order”.
* **flood\_fill** – a recursive function that fills in a circular region in the board with a specified fill character. The center location (r,c) and radius of the circular region are input parameters. This method should only overwrite Empty characters in the board, and should stop filling when non-Empty characters are encountered. This way, the circular fill will simulate an explosion that does not penetrate walls.

Your second task will be to create a simple main program that lets the user create a simple game environment by building walls with line segments, placing creatures at selected locations, and simulating explosions with circular bomb blasts. The purpose of the main program is NOT to create a working game, but simply to test each of the methods in the GameBoard class. When you run your program, you can use the read\_board and write\_board methods to initialize the board at the start of the program, and save your final board after testing various methods above.

**2. Design:**

Your primary design tasks are: (1) to select the data types for all of the private variables in the GameBoard class, (2) to specify the parameters for all of the GameBoard methods, and (3) to work out the algorithms you will use to implement all of the methods. We will be talking about the algorithm for flood\_fill in class. For the examples of good class designs, you are encouraged to look at the textbook, the class notes, and sample source code for inspiration.

**3. Implementation:**

This time, you are starting with an “empty page” when implementing your GameBoard class. It would probably be a good idea to copy an existing example from your last project or the source directory to get the basic GameBoard.h and GameBoard.cpp and main.cpp layout. Then delete the information you do not need, and start adding private variables and methods one at a time. It is always a good idea to work incrementally, writing comments, adding code, compiling, debugging, a little bit at a time. Once you have the methods implemented, you can add calls to the methods to your main program to complete your project.

Remember to use good programming style when creating your program. Choose good names for variables and constants, use proper indenting for loops and conditionals, and type clear comments while you are writing the code. Also, be sure to save backup copies of your program somewhere safe. Otherwise, you may end up retyping your whole program if something goes wrong.

**4. Testing:**

Test your program to check that it operates correctly for all of the requirements listed above. Also check for the error handling capabilities of the code. Try your program with several input values, and save your testing output in text files for inclusion in your project report.

**5. Documentation:**

When you have completed your C++ program, write a short report using the project report template describing what the objectives were, what you did, and the status of the program. Does it work properly for all test cases? Are there any known problems? Save this report to be submitted electronically.

**6. Midpoint Project Submission:**

To encourage students to get an early start on their programming project, students are required to upload into Blackboard a partial solution to their programming project on the midpoint due date shown above. The program does not need to be complete, but it must compile and perform some of the tasks listed above. This midpoint solution is worth 10% of your final grade on the project.

**7. Project Submission:**

In this class, we will be using electronic project submission to make sure that all students hand their programming projects and labs on time, and to perform automatic plagiarism analysis of all programs that are submitted.

When you have completed the tasks above go to Blackboard to upload your documentation (a single docx or pdf file), and all of your C++ program files. Do NOT upload an executable version of your program.

The dates on your electronic submission will be used to verify that you met the due date above. All late projects will receive reduced credit:

* 10% off if less than 1 day late,
* 20% off if less than 2 days late,
* 30% off if less than 3 days late,
* no credit if more than 3 days late.

You will receive partial credit for all programs that compile even if they do not meet all program requirements, so handing projects in on time is highly recommended.

**8. Academic Honesty Statement:**

Students are expected to submit their own work on all programming projects, unless group projects have been explicitly assigned. Students are NOT allowed to distribute code to each other, or copy code from another individual or website. Students ARE allowed to use any materials on the class website, or in the textbook, or ask the instructor and/or GTAs for assistance.

This course will be using highly effective program comparison software to calculate the similarity of all programs to each other, and to homework assignments from previous semesters. Please do not be tempted to plagiarize from another student.

Violations of the policies above will be reported to the Provost's office and may result in a ZERO on the programming project, an F in the class, or suspension from the university, depending on the severity of the violation and any history of prior violations.