Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Spring 2019

Homework 6 – Battleship Game with Object Sharing

Due: 26/04/2019, Friday 21:00

PLEASE NOTE:
Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You can NOT collaborate with your friends and discuss solutions. You have to write down the code on your own. Plagiarism will not be tolerated!

Introduction

In this homework, you are asked to implement a simple board game using the *object sharing* concept of C++. The main function of the implementation, which includes the game implementation using classes, is given to you as well as the .exe file to try out the game. You are expected to write two classes: *Board* class and *Player* class. We are going to explain these classes in more details in the following sections. Before that, we start with a general description of the game.

The Game

This game is a version of the well-known two-player game called *Battleship*. In our version, the game board is a 6×6 integer matrix (of `integer` type). The first half of the board belongs to player id 1 and the other half belongs to player id 2, which is divided column-wise. See Figure 1 for the areas owned by the players.

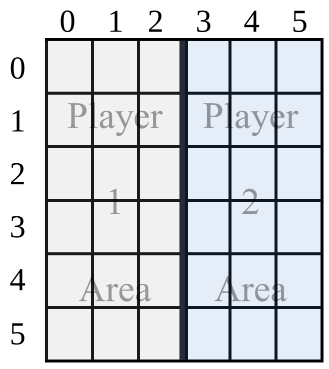

Figure 1. The battlefield and the areas of the players

At the start of the game, each player places their ships on the area allocated to them. This game includes two types of ships:

Ship ID 1: Small ship with size 1 x 2 or 2 x 1 (two horizontal or vertical neighboring cells) .
Ship ID 2: Large ship with size 1 x 3 or 3 x 1 (three horizontal or vertical neighboring cells).

As in the original game, each ship can be placed once, and the game does not start until all the ships are placed on the board. Moreover, the ships cannot be placed on top of each other (wholly or partially).

After placing all the ships, the game starts. The goal of the game is to destroy all of your opponent's ships before your ships are destroyed. To destroy a ship, a player needs to bomb all of the coordinates that the ship contains.

After the game starts, players take turns to make guesses. Each guess targets a coordinate in your opponent’s area of the board. If the guess is successful, the board is updated and players are informed via messages. Otherwise no update is made but still players are informed. As inferred, to win the game a player needs to destroy both of the opponent's ships before his/her ships are destroyed.

Program Flow

First, each user places two types of ships on the board space allocated for them. There must be a number of checks in order to validate a correct placement. Then players take turns at guessing the coordinates of the enemy ships, where each valid guess will end up either with a *hit* or a *miss*. Once a player succeeds at hitting all the coordinates that contains a ship piece, game ends and the respective player wins the game. While displaying the board, `0` will be used to display empty cells, `1` for the cells that contain ship pieces, and `-1` for the bombed (hit) ship locations.

Actually, this logic has been partially implemented in main and it is provided to you.

All integer inputs (for the row and column values) are assumed to be entered as integers; if you enter a non-integer while trying the provided .exe file, the program may get into infinite loop. Thus, please do not enter non-integer inputs for row and column values.

The `Board` Class

The `Board` class will be used to create and manipulate a board on which the game will be played. A board is represented by a built-in matrix, which is a private data member of the class. The matrix will have a fixed number of rows and columns. You can define this private data member as shown below.

```
int theBoard[6][6];
```

Now, we will give the constructor and some of the member function explanations of the `Board` class.

Default Constructor: You need to initialize all of the matrix elements to 0, which indicates that the cells are vacant.

displayBoard: The displayBoard function does not take any parameters. It only displays the current state of the board. You need to display the game board exactly as you will see when you run the executable file that comes with this homework pack. Here, we used `setw` I/O manipulator is used for tidy output; you are recommended to use the same.

putShip: This function returns `bool`. It will return `true` if the user successfully places a ship; otherwise, it returns `false`. It is required from the developer (i.e. you) to check whether (a) the ship's shape is correct, (b) the coordinates are valid (within the player's side of the board), and (c) all of the cells that the ship will be placed are empty, meaning no two ships coincide. This function will take *player id*, *ship id*, *rowStart*, *colStart*, *rowEnd* and *colEnd* as integer parameters. Note that *colStart* can be bigger than *colEnd* and/or *rowStart* can be bigger than *rowEnd*, meaning that the ship can be placed in any orientation. Actually the orientation of the ships does not have any effect in the game, but while placing the ship, we have this flexibility. A ship is represented by having 1 as the corresponding cell values.

makeAHit: This function also returns `bool`. It takes *row* and *column* as two integer parameters. This function is for guessing whether a particular cell contains a ship piece. The functions return `true` if the guess is valid; otherwise returns `false`. In other words, it returns `true` if the parameter coordinate contains a ship piece that has not been previously hit. If the guess hits a ship, then that cell is marked by -1 indicating that the hit is successful. In this function, you will not make any checks about the validity of the coordinate values; you can assume that the parameter coordinates are valid.

checkIfPlayerWon: This function returns `bool`. It takes *player id* as parameter and returns `true` if all the locations which contain a ship piece are hit (i.e. value is -1) in the area of the opponent of *player id*. Otherwise, returns `false`.

Two of the above functions (namely `displayBoard` and `putShip`) are explicitly called in the game implementation given to you in main.cpp. However, the others are not directly called in main.cpp; but they need to be used by the `Player` class member functions. Since the use of friend functions and friend classes are **not** allowed in this homework, in the implementation of the `Player` class, you will need to use them to manipulate the shared board object. You do **not** need other functions (especially accessors/mutators) for the `Board` class; thus please do not add them.

The `Player` Class

The `Player` class will be used to manage the players of the game. There will be two players playing on the same board in a game. Thus, player objects must share a Board object using *object sharing* concept and principles of C++ as we have seen in the lectures. We have seen two different methods for object sharing in the course; due to our main function implementation, which is provided to you, you must use the reference variable method.

In addition to the shared Board object, the player class should also keep its id in an integer type of private data member. Now, we will give constructor and member function explanations of the `Player` class.

Constructor: The constructor of the *Player* class takes two parameters, which are the `Board` object that will be played on and *player id* (1 or 2). These parameters are used to initialize the corresponding private data members.

guess: This is a function with `bool` return type. As you can see in main.cpp, it is called in each player's turn after the players place all their ships. It takes two integer parameters for the *row* and the *column* of the location to be bombed on the board. This function will directly call the `makeAHit` function of the `Board` class through its private board object and return its result.

wins: This function also returns `bool` and checks if the player has won the game. It does not take any parameters. It will call `checkIfPlayerWon` function of the `Board` class to perform the check and will return its result.

This member functions are sufficient; you do not need any other member functions and please do not add more functions.

Using Object Sharing Principles and Object Oriented Design

In your program, the *Board* object must be shared by the *Player* objects. For this object sharing, you have to employ the method that uses reference variables.

It should be clear that you will write two classes for `Board` and `Player`. You need to analyze the requirements carefully and make a good object-oriented design for these classes. In this context, you have to determine the data members and design/implement member functions of each class correctly. We will evaluate your object oriented design as well. Moreover, you are not allowed to use friend class or friend functions in your design (i.e. you are not allowed to use the `friend` keyword anywhere in your code). Our aim by this restriction is not to make your life miserable, but to enforce you to proper object oriented design and implementation.

Provided files

battleship.exe: This is the executable file of our implementation for the Battleship board game. In this homework, we do not provide any sample runs due to the interactivity of the game. Instead we provide this executable so that you can try and understand the requirements.

main.cpp: This file contains the `main` function and the related code, which contain the game implementation by using related class functions. In this homework, our aim is to reinforce object oriented design capabilities; thus, we did not want you to deal with the class usage, but focus on their design and implementation. Please examine this provided file in order to understand how the classes are used and how the game is played. We will test your codes with this main.cpp with different inputs. You are not allowed to make any modifications in this file (of course, you can edit the file to add `#include`s, etc. to the beginning of the file); you have to create and use other files for class definitions and implementations.

Do not forget to submit all .h and .cpp files (including the classes' .h and .cpp files, and the main .cpp file) in addition to other project related files.

Please see the previous homework specifications for the other important rules and the submission guidelines

Good Luck!
Taha Atahan Akyıldız, Albert Levi