CS 185
Programming Assignment 7

Details

This assignment will give you some practice with object-oriented programming (classes, objects, constructors, destructors, etc.), namespaces, references, dynamic memory allocation, and 2-dimensional arrays. It will also give you a chance to learn about interfaces and how to read source code. The task is to implement a very simple version of the popular board game *Battleship*. The player will place boats in an "ocean" that you create and will attempt to sink each one by taking shots into the ocean.

0	0	0	0	0	0	0	0
0	0	0	0	0	**2**	0	0
0	0	0	0	0	**2**	0	0
0	**1**	**1**	**1**	**1**	**2**	0	0
0	0	0	0	0	**2**	0	0
3	**3**	**3**	**3**	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

An 8x8 board with 3 boats placed.

−1	−1	−1	0	−1	−1	0	−1
−1	−1	−1	−1	−1	**102**	−1	0
−1	0	−1	−1	−1	**102**	−1	−1
−1	**101**	**101**	**101**	**101**	**102**	−1	−1
−1	−1	0	−1	−1	**102**	−1	−1
103	**103**	**103**	**103**	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	−1	−1

The board after sinking all 3 boats.

Instead of simply hard-coding the size of the board at 10x10, we are going to allow the player to specify the dimensions of the board. (The board is the ocean.) You should be able to handle boards of any rectangular size (square and non-square). You are also going to allow the player to specify the number of boats to place in the ocean. The only limit to the number of boats is the size of the ocean, since boats will not be allowed to overlap or leave the ocean. (Contrary to what Ferdinand Magellan's crew claimed, the world is flat, and if you go too far, you will fall off the end of it.)

In order to keep the implementation simple, all of the boats will be the same size. Also, only the player will be taking shots, meaning that the computer will not be trying to find the player's boats. We are studying C++ in this course, not artificial intelligence. (However, after completing this assignment, you will have most everything you need to create a more sophisticated game.)

Like all programs, there are many ways to implement this. For this assignment, you are given some header files to use as a starting point. These files give you the layout of the solution. The interface to the game is included in a header file named WarBoats.h. This file contains all of the information that the client (the player) needs. You are also given a file named Ocean.h, which defines what an *Ocean* is.

All of your implementation will be placed in Ocean.cpp and Ocean.h, and these will be the only two files that you will submit. You are not allowed to include any other files in Ocean.cpp. There are a couple of files included already, but you will not need any others.

Methods Details

There are only 5 methods that you need to write for this assignment, and three of them are fairly simple.

Methods listed in the class
`Ocean(int num_boats_, int x_quadrants_, int y_quadrants_);`
Constructor - The client calls this to create an ocean (game board). Client specifies the dimensions of the ocean and the number of boats that will be placed in it. All private fields are initialized. No return value.
`~Ocean(void);`
Destructor - This method is responsible for deallocating anything that was allocated in the constructor. No return value.
`BoatPlacement PlaceBoat(const Boat& boat_);`
The client calls this to place boats in the ocean. The client will create a boat and pass it to the method. The method must ensure that the boat will "fit" in the ocean. Do this by checking the location where it is to be placed, the orientation (horizontal/vertical), and whether or not it will overlap with another boat or stick outside of the ocean. The return value indicates whether or not the boat could be placed in the ocean.
`ShotResult TakeShot(const Point &coordinate_);`
Client calls this in an attempt to hit one of the boats. The coordinate parameter indicates where the client is attempting to strike. There are several possible results: Hit, Miss, Sunk, Duplicate, or Illegal. Hit, Miss, and Duplicate are obvious. Sunk is returned when a shot hits the last undamaged part of a boat. Sunk implies Hit. Illegal is any coordinate that is outside of the ocean (e.g. x/y less than 0 or outside the range).
`ShotStats GetShotStats();`
Client calls this to get the current status of the game.

To get a feel for how these methods are supposed to work, you simply need to look at the code that is provided in the sample driver. This is the best way to see how they are used. All of these methods are called in the sample driver, some of them many times. The client (driver) will typically call *the Ocean* constructor and destructor only once, at the beginning and end of the game, respectively. The driver will call *PlaceBoat* once for each boat to place. However, if the driver tries to place a boat in an illegal location, you will return a value indicating it's an illegal location and the driver will try again with another location. Once the boats are placed, the player (driver) makes repeated calls to *TakeShot*, which is the primary action during the game. The player can stop taking shots when all of the boats have been sunk.

The only methods that require non-trivial code are *PlaceBoat* and *TakeShot*. The *PlaceBoat* method requires you to make sure that a boat can be legally placed at the specified position. You'll have to have some logic that checks to make sure the boat placement is legal. The *TakeShot* method requires you to validate the coordinates and then to determine the type of shot (e.g. hit, miss, duplicate, etc.)

Remember to use **const** wherever appropriate. This includes both method parameters as well as methods (member functions).

Comments

In this and future assignments, you are required to include:

- **A file header comment in every piece of source file. The format is shown in the "Comments.cpp" file given to you in the beginning of the semester and should be present at the very top of all your code.**
- **Function header for each function you create. The format is shown in the "Comments.cpp" file given to you in the beginning of the semester and should be present at the top of every function.**
- **Inline commenting for your code.**

What to submit

You must submit the header file and implementation file ***(Ocean.h, Ocean.cpp)*** in a single .zip file (go to the class page on moodle and you will find the assignment submit link). ***Do not submit any other files than the ones listed.***

Source files	Description
Ocean.cpp	The implementation file. All implementation for the methods goes here. You must document the file (file header comment) and methods (function header comments).
Ocean.h	The header file. You will need to modify this. **Absolutely NO implementation is permitted in this file.**

If you've forgotten how to submit files, the details are posted in the syllabus and in the assignment guidelines document. Failure to follow the instructions will result in a poor score on the assignment (and possibly a zero).

Usual stuff

Your code must compile cleanly (no errors and no warnings) to receive full credit. You should do a "test run" by extracting the files into a folder and verifying that you can compile and execute what you have submitted (because that's what I'm going to do.)

Special note:

The due date/time posted is the positively latest you are allowed to submit your code. Since the assignments can easily be completed well before the deadline, you should strive to turn it in as early as possible. If you wait until the deadline, and you encounter unforeseen circumstances (like being sick, or your car breaking down, or something else), you may not have any way to submit the assignment on time. Moral: **Don't wait until the last day to do your homework.**