ECS40 Fall 2014 2014-11-14

Homework 3 due Tue 2014-11-25 at 23:50

Use the handin directory hw3 to submit your work

Battleship game – part II

Description

In this assignment (HW3), you will complete the object-oriented implementation of the battleship board game initiated in HW2.

Battleship game

The basic description of the game, including types of ships and valid configurations, was given in HW2. The game will be implemented for a single user playing against the computer. The game is interactive and reads input from **`stdin`** and writes output to **`stdout`**. It can also read input from a file using a < redirection.

Ten ships (one aircraft carrier, two battleships, three cruisers and four destroyers) are placed on the board. The board columns are labeled by a letter and the rows by a number, with letters from 'a' to 'j' and numbers from 0 to 9.

The game starts with the program reading the arrangement of ships from the configuration file (provided) and placing the ships on the board. It is assumed that the file contains a valid arrangement, i.e. there are no overlapping or adjacent ships. The player then enters the coordinates of a cell to try to hit a ship. The program responds by announcing if a ship is hit. If all cells occupied by a ship are hit, the program prints a message stating that that ship is sunk. After each attempt, the program prints the current state by showing the board with all successful attempts marked by "*" and unsuccessful attempts marked by "x". The game ends when all ships are sunk, or when the player stops input (by entering Ctrl-D, or by reaching the end of file if attempts are read from a file). The player's score is the number of times a cell was hit (i.e. lower scores are better).

HW3 Assignment

In HW3, you are given the file **`battleship.cpp`** (which contains the **`main`** function), and the file **`Board.h`**, which contains the declarations related to the class **`Board`**. Your task is to implement the class **`Board`** by writing the file **`Board.cpp`**. You should not modify the files **`Board.h`** and **`battleship.cpp`**. The file **`Ship.h`** is the same as in HW2 and should not be modified. You may update the file **`Ship.cpp`** that you used in HW2. All files are provided at http://web.cs.ucdavis.edu/~fgygi/ecs40/homework/hw3

Your files should compile without warning. You should be able to build the executable **`battleship`** using the command

```
$ make
```

You will use the **battleship** executable and the input and output test files provided on the web site to check the functionality of your classes and verify that your program reproduces the test output *exactly*. Note that these test files do not check all the functionality and you are responsible for verifying your implementation. Other test files will also be used when grading your implementation. The **battleship** program reads input from **stdin** and writes to **stdout**. It can be used e.g. as

```
$ ./battleship < test1.in
```

Specification of the Board class

The **Board** class is responsible for managing the Ship objects during the game.

public members

Board
Constructor. Must initialize members appropriately.

void addShip(char type, int x1, int y1, int x2, int y2)
Add a ship to the board. If the board already contains ten ships, the new ship should be ignored.

void print(void)
This function prints the current state of the board using the format shown in the example below.

void hit(char c, int i)
This function is called when the player attempts to hit a cell defined by char c and int i.
The function must determine if a ship is hit and update the ship's level and board status accordingly. It uses the private function **shipAt**. The **hit** function must check the validity of the character and integer arguments and throw an exception of type **invalid_argument** if the data is not valid.

int level(void)
This function should return the sum of the levels of all ships on the board.

private members

vector<Ship *> shipList
A vector of base pointers to store pointers to the ships

char score[10][10]
An array of **char** storing the current state of each cell of the board, with char **'x'** and **'*'** representing unsuccessful and successful attempts respectively, and **' '** (space) in cells that were not hit. The character **score[0][0]** corresponds to the upper left corner of the board, i.e. cell **a0** .

Ship *shipAt(int x, int y)
If a ship occupies the cell (x,y) this function returns a pointer to that ship. Otherwise it returns a null pointer.

Description of the battleship program

The **battleship.cpp** program (provided) is started using the command line:

```
$ ./battleship
```

The following two steps are then repeated until the game is completed:

1) The player enters the coordinates of a cell by typing two letters (such as **c3** or **h7** or **i0**) followed by the Enter key. The program reads the coordinates and determines if a ship was hit. If all cells occupied by a ship are hit, the program prints:

```
<ship> sunk
```

where **<ship>** is replaced by the type of the ship being hit, i.e. **AircraftCarrier**, **Battleship**, **Cruiser** or **Destroyer** .

2) The program prints the current state of the board showing all successful and unsuccessful attempts so far (using '**x**' for unsuccessful attempts and '*****' where a ship was hit).

After completing step 2), the program checks whether all ships are sunk. If all ships are sunk, the game is completed and the program prints

```
Game over! Your score is <number of attempts>
```

where **<number of attempts>** is replaced by the total number of attempts.
If there are still ships afloat, the program goes back to step 1) and reads input.

The following lines show the format to be used when printing the current state of the board:

```
  a b c d e f g h i j
 +--------------------+
0|     *    *         |
1|          x         |
2|                    |
3|                  x |
4|       x            x|
5|*                   |
6|               x    |
7|                    |
8|     x            * *|
9|   *      *         |
 +--------------------+
```

Error processing and special cases

If the player enters incorrect cell parameters (letter not in [a-j] or number not in [0-9]), the program should print

`Error: invalid input`

(and no other messages) and continue reading input until correct input is entered.

If the player enters an attempt that has already been entered before, it should be ignored (i.e. it does not further lower the level of a ship) but note that the total number of attempts is incremented in battleship.cpp.

Initial configuration

The arrangement of ships is read from a configuration file named **`battleship.conf`** located in the current directory. The file contains the type and position of each ship. Each ship is specified on a separate line. The format of a line is

`<symbol> x1 y1 x2 y2`

where **`<symbol>`** is **`A,B,C`** or **`D`** (for **`AircraftCarrier`**, **`Battleship`**, **`Cruiser`** or **`Destroyer`**), and **`x1,y1,x2,y2`** are the positions of the two cells at both ends of the ship as in HW2. For example, the aircraft carrier appearing in the grid shown in HW2 would be specified as

`A 2 0 6 0`

Note that the y coordinate is increasing in the downward direction.

It is assumed that the configurations given in the configuration file are correct, i.e. that ships do not intersect or touch, and that they are within the limits of the board.

Submission

Create a tar file named **`hw3.tar`** containing the files **`battleship.cpp Ship.h Ship.cpp Board.h Board.cpp`** and **`Makefile`**. Do not compress the tar file. The Makefile should include the necessary definition to compile C++ files with the **`-Wall`** option. Include your name and Student ID in a comment at the top of each file that you wrote (i.e. not in the files provided). Your file **`Ship.cpp`** may differ from the one you submitted for HW2. Submit your project using:

`$ handin cs40 hw3 hw3.tar`