ECS40 Fall 2014 2014-10-31

Homework 2 due Thu 2014-11-13 at 23:50

Use the handin directory hw2 to submit your work

Battleship game – part I

Description

In this assignment (HW2) and in the next assignment (HW3), you will write an object-oriented implementation of the battleship board game.

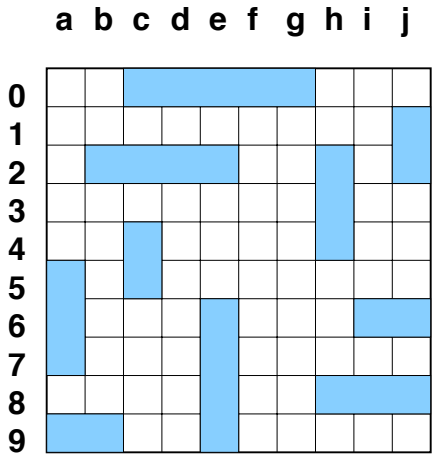
Battleship game

The game is played on a board consisting of 100 cells arranged as a 10x10 square. Ten ships are placed on the board. There are four kinds of ships. Each ship occupies a number of cells that depends on the size of the ship. The ship types are

1) Aircraft carrier, size=5
2) Battleship, size=4
3) Cruiser, size=3
4) Destroyer, size=2

A ship occupies contiguous cells arranged as a rectangle of width=1 and height=size, or width=size and height=1, i.e. the orientation of the rectangle can be horizontal or vertical. Ships cannot overlap and cannot touch each other (either sideways or diagonally).

The following figure shows a valid arrangement of ships on the board.



HW2 Assignment

The goal of HW2 is to implement classes representing ships. These classes will be later used in HW3 to implement the full battleship board game.

The class **`Ship`** is an abstract base class from which four classes are derived: **`AircraftCarrier`**, **`BattleShip`**, **`Cruiser`** and **`Destroyer`**. A detailed specification of the class **`Ship`** is given below. You are given the header file **`Ship.h`** (at http://web.cs.ucdavis.edu/~fgygi/ecs40/homework/hw2). Your task is to write the implementation file **`Ship.cpp`** containing the implementation of all classes. You are not allowed to modify the file **`Ship.h`** (Do not insert your SID in it since we will use the cmp command to check that the file is the same as the original). Your class implementation file **`Ship.cpp`** should compile without warning using the command

```
$ g++ -Wall –c Ship.cpp
```

You are also given a program **`testShip.cpp`** and a **`Makefile`** (same web site as above). These two files should not be modified. The **`testShip.cpp`** program tests the functionality of the **`Ship`** class. You should be able to build the executable **`testShip`** using the command

```
$ make
```

You will use the **`testShip`** 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*. Use the diff command to compare your output file with the output file provided. 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 **`testShip`** program reads input from **`stdin`** and writes to **`stdout`**. It can be used e.g. as

```
$ ./testShip < test1.in
```

Specification of the Ship class

We describe here the detailed specification of the **`Ship`** class. The location of a ship is stored in two pairs of integer variables (x1,y1) and (x2,y2) which specify the coordinates of the cells located at both ends of the ship on the cell grid. The class includes a variable used to store the current "level" of the ship (stored in the protected variable **`lev`**). The level will be used to keep track of the number of times the ship has been hit during the game. When the ship is created, the level is equal to the size of the ship. During the course of the game (implemented in HW3) it will be decreased by one every time the ship is hit.

Study carefully the file **`Ship.h`**. Some members are public, some protected and some private. Some members are virtual. Make sure you understand why. Study the file **`testShip.cpp`** in order to understand how the classes will be used.

Public members

`virtual const char *name(void)`
This function should return the name of the class instance as a C string, i.e. **`AircraftCarrier`**, **`BattleShip`**, **`Cruiser`** or **`Destroyer`**.

`virtual int size(void)`
This function should return the size of the ship according to the sizes in the table given above.

`int getX(int i)`
This function returns the x coordinate of the ith cell of the ship. The value of i is in [0,size-1] where size is the size of the ship. **`getX(0)`** should return x1 and **`getX(size()-1)`** should return x2. It is assumed that the argument i is valid i.e. it is in [0, size-1]. Note that x1 may be larger than x2.

`int getY(int i)`
Same as **`getX`** but for the y coordinate.

`void print(void)`
This function prints information about the location of the ship by showing the cells occupied by the ship. It should use the following format:

`<name> from (<x1>,<y1>) to (<x2>,<y2>)`

For example:

`AircraftCarrier from (0,0) to (0,4)`

`bool includes(int x, int y)`
This function should return **`true`** if the cell (x,y) is occupied by the ship, and **`false`** otherwise.

`int level(void)`
This function should return the current value of the protected variable **`lev`**

`void decreaseLevel(void)`
This function should decrease the variable **`lev`** by one. If the value of **`lev`** is zero, the value should be unchanged (i.e. **`lev`** should not become negative)

`static Ship *makeShip(char ch, int x1, int y1, int x2, int y2)`
This function is a "ship factory". It is used to generate instances of ships. It creates an instance of the appropriate kind of ship (specified by the character **`ch`**, with values **`'A'`**, **`'B'`**, **`'C'`**, **`'D'`**) at the given position using the operator **`new`**, and returns a pointer to the instance. If the character **`ch`** is not one of the allowed letters, the function should throw an **`invalid_argument`** exception.

protected members

`void setPos(int x1, int y1, int x2, int y2)`
This function is used to set the values of the private variables x1,y1,x2,y2. It should check the validity of the combination (x1,y1) and (x2,y2) using the **`checkConfig`** function and throw an **`invalid_argument`** exception if the combination is invalid.

`int lev`
This variable stores the current level of the ship.

private members

`bool checkConfig(int x1, int y1, int x2, int y2)`
This function checks that the combination (x1,y1) (x2,y2) is valid, i.e. the two cells (x1,y1) and (x2,y2) are either on the same row or on the same column. It also checks if the distance between the two cells is compatible with the size of the current instance of ship calling the function. For example, the pair (2,3) (2,5) is valid for a **`Cruiser`** since it includes a total of three cells. The function should return **`true`** if the configuration is valid and **`false`** otherwise. Hint: use the **`size()`** member function to get the correct size.

`int x1,y1,x2,y2`
The coordinates of the cells located at both ends of the ship. The ship occupies these cells and all cells located in between.

Implementation advice
Use the function **`setPos`** in the constructors of the derived classes.

Submission
Create a tar file named **`hw2.tar`** containing the files **`Ship.h, Ship.cpp, testShip.cpp`** and **`Makefile`**. Do not compress the tar file. The files **`Ship.h, testShip.cpp`** and **`Makefile`** must be identical to the files provided. Include your name and Student ID in a comment at the top of the **`Ship.cpp`** file. Submit your project using:

```
$ handin cs40 hw2 hw2.tar
```