**CENG 201**

**Object-Oriented Programming**

**HOMEWORK 4**

**Due Date: 04.01.2015 23:59**

In this homework, you will required to develop the software for playing the game ***AmiralBatti***. The game is played on a grid where the grid is typically a square and the squares in the grid are identified by two numbers for the grid positions. The positions are numbered from 0 to n for a nxn grid. The top left corner will be (0,0). For instance, for a 10x10 grid valid cells are (0,0)(0,1)(0,2) ....(0,9)(1,0)(1,1).....(1,9)....(9,9) which is totally 100 cells.

In the following lines, the rules of the game and the requirements you need to implement are described. Please go over each sentence and be sure to understand what is requested before starting implementation.

1. Before game begins, a number of ships must be arranged secretly on the grid. Each ship occupies some fixed number of consecutive squares on the grid arranged either horizontally or vertically. There are 4 different types of ships each with different sizes. The ships cannot overlap (at most one ship can occupy any given square in the grid). The sizes of the ships are given below:

|  |  |
| --- | --- |
| **Ship** | **Size** |
| Battleship | 4 |
| Destroyer | 3 |
| Submarine | 2 |
| Boat | 1 |

1. There are 3 types of equipment that the player can have in order to shot the ships. The properties of the equipment are specified in the table below:

|  |  |  |
| --- | --- | --- |
| **Equipment** | **Property** | **Example** |
| Gun Shot | Shots 1 square at the specified position | Gun Shot made to (1,2)-->shots the position (1,2) |
| Hand Bomb | Shots 3 squares; horizontally left, right of the specified cell and the cell itself. | Hand Bomb sent to (1,2)-->shots the positions (1,1),(1,2), and (1,3) |
| Rocket | Shots 5 squares; horizontally left, right of the specified cell, vertically up and down of the specified cell and the cell itself. | Rocket sent to (1,2)-->shots the positions (1,1),(1,2),(1,3),(0,2) and (2,2) |

1. The game has three levels of difficulty (easy, normal and hard) with the following rules and properties:

|  |  |  |  |
| --- | --- | --- | --- |
| **Difficulty Level** | **Grid Size** | **Ships** | **Equipment of the Player** |
| Easy | 10x10 | * 1 boat, * 1 submarine, * 1 destroyer | 1 rocket, 2 hand bombs and 7 gun shots |
| Normal | 15x15 | * 2 boats, * 1 submarine, * 1 destroyer * 1 carrier | 1 rocket, 3 hand bombs and 10 gun shots |
| Hard | 20x20 | * 2 boats, * 1 submarine, * 1 destroyer * 1 carrier | 1 rocket, 4 hand bombs and 12 gun shots |

* Therefore, for instance, difficulty level “easy” sets a grid of 10x10 with one ship of boat, submarine and destroyer. There will be no battleships. The user has 1 rocket, 2 hand bombs and 7 gun shots in order to shot all the ships placed in the grid.
* The difficulty level is easy for a new player.
* After a player wins 2 games in easy level, the player starts playing in normal level
* After a player wins 3 games in normal level, the player starts playing in hard level

1. Your application should keep a file named "**AmiralBattiInfo.txt**" which you write the information about the users as follows (You should put the file in the project folder so that while testing your application, it should not lead to any error due to accessing it):

* Each user will have a unique username that will be set before starting to play, therefore when the game begins, your application should ask to enter username and login, or create a new user.
* If username is entered to login, you will check the username from "AmiralBattiInfo" file.
* The file will have the following format:

<username> <mode>(<level>)

There will be one line for each user. An example of the file is specified below. In this example, we observe that two users have played the game. Upto now, Ayse completed 1 game in easy mode, Ali completed 3 games in normal mode.

**Ayse easy(1)**

**Ali normal(3)**

* Therefore for each user, the user's level together with how many games she wins at that level will be kept.
* The game then starts with the corresponding level.
* For instance, for Ayse, game starts at easy mode, and after Ayse wins one more game in easy mode, she starts playing in normal mode, since she will totally have winned 2 games in easy mode.
* If a user enters a username that does not exist in the "AmiralBattiInfo.txt" file, you should give "Wrong Username" message and display the game menu again.
* The menu displayed before starting the game will be as follows:

**GAME MENU**

**1. Login**

**2. Create new user**

* if 1 is pressed, then you will request a username. After a valid username is entered, you will start the game. If not you will redisplay the menu.
* If 2 is pressed, you will create a new user with the username entered and you will start the game in easy mode.

**How to Play the Game**

* In order to start the game, your application should first place ships in the grid correctly. After the ships have been positioned, the game proceeds in a series of turns. In each turn, the player attempts to “hit” a square from the grid by entering two numbers which corresponds to a cell of the grid.
* If there exists a ship at the specified position, it is a hit.
* When all the parts of a ship is destroyed, then that ship is sunk.
* The player does not know where the ships are.
* When a ship is hit but not sunk, the program does **not** provide any information about what kind of a ship was hit.
* However, when a ship is hit and sinks, that is all parts of the ship is destroyed, the program prints out a message "You just sank a *XXX*." where XXX is the ship type.
* After each shot, the computer redisplays the grid with the new information. How to display the grid is described below.
* A ship is "sunk" when every square of the ship has been hit. Thus, it takes four hits to sink a battleship, three to sink a destroyer, two for a submarine, and one for a boat.
* The main objective of the game is to sink every ship in the grid with the available equipment. For instance, in order to win a game in easy mode, user has to shot all ships with 1 rocket, 2 hand bombs and 7 gun shots.
* Whenever a shot is not valid due to the range of the grid, an error message "Invalid Hit" should be given. For instance, (0,3) will not be a valid shot with a rocket. In that a case a new hit from the user will be requested.
* When all ships have been sunk successfully, the program prints out a message that the game is over successfully, and asks the user whether to continue in the correct level or exit.
* If user loses the game, the program will again print out a message that the game is over but this time unsuccessfully, and asks the user whether to continue in the correct level or exit.
* If user chooses to exit, the "AmiralBattiInfo.txt" should be updated accordingly if user wins games. Therefore you should keep how many games the user won and her current level. For instance if Ayse completed Easy mode and played 1 game successfully in normal mode, you should store in the file "Ayse normal(1)" for the player Ayse.

**How to Display the Grid**

To aid the user, row numbers should be displayed along the left, and column numbers should be displayed along the top for the grid. You should use 'S' to indicate a location that you have fired upon and hit a ship, '-' to indicate a location that you have fired upon and found nothing there, 'x' to indication location containing a sunken ship (batırılan gemi), and '.' to indicate a location that you have never fired upon.  Below a sample grid is shown:

0 1 2 3 4 5 6 7 8 9

0 . . . . . . . . . .

1 . . - . . . . . . .

2 . x x x . . . . . .

3 . . - . - . . . . .

4 . . . . . . . . . .

5 . - S S . . . . . .

6 . . . . . . . . . .

7 . . . . . . . . . .

8 . . . . . . . . . .

9 . . . . . . . . . .

Here in this sample, the player shots a destroyer at position (2,1)(2,2,)(2,3). She may have shot it with a rocket since the up and down cells of (2,2) have also been shot. But, since there is no ship there, it is not a hit for the player. Therefore, if you use a rocket or bomb, only the parts that hit a ship will be displayed with x(for sunk) or S(for hit but the ship not sunk). The other cells should be displayed with '-'.

REQUIREMENTS:

1. Be sure to implement everything requested. So go over the game specifications several times and tick the item that you have completed implementing to keep track of the parts of the assignment you have completed.
2. Write good code, use inheritance and polymorphism whenever you can.
3. Prepare a one page report in which you describe your classes and how you use inheritance and polymorphism concepts in your solution.
4. Send a one .rar file including both your solution and report.
5. The homework will be done in pairs (2 people). Write the names of the participants in the first line of the report and in your email.