CSCI 201L Assignment #2
Spring 2015
2.5% of course grade

Exploring GUI: An exposure to Java AWT/Swing with Battleship

Overview

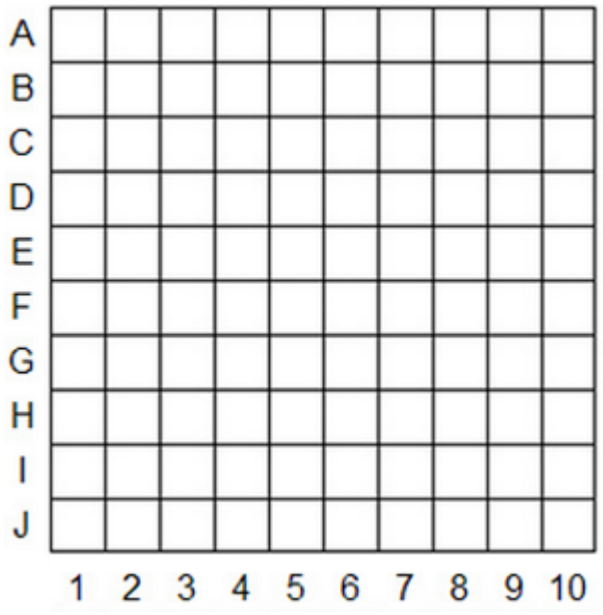
This assignment will have you create a one-player battleship game. The main goal is to expose you to working with Java's LayoutManager classes and graphical elements. You will need to manage data about the game behind the scenes while displaying the relevant information to the user. You will need to be patient and persistent to get the graphical elements to behave the way you would like. However, with more practice it will become easier and faster to develop programs involving graphical elements.

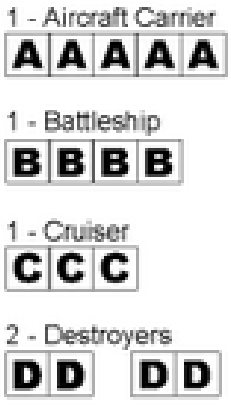
Introduction

If you've never heard of battleship, don't worry - it is explained below. However, it may also be a good idea to Google it and check it out. You will be creating a one-player game of battleship. The game will consist of a 10x10 grid with labels A-J and 1-10. The program you make will read in data about ships from a file, and place them onto the grid that are hidden to the player. The player must guess coordinates until all of the ships have been sunk. Once the player wins, their score will be recorded, which will be the number of guesses. In addition to this, a high-score table will be displayed to the player and update after the player wins and beats another ranked player.

Background

Battleship is a two-player game where each player has a 10x10 grid playing field. This field represents a sea in which players place their ships. One aircraft carrier, one battleship, one cruiser, and two destroyers. Each type of ship is of a different size, but all ships must be placed vertically or horizontally – there is no diagonal placement of ships. The players take turns guessing grid locations of their opponent's ships. Once all grid spaces that are occupied by a ship have been guessed, the ship sinks. The first player to sink all of the opponent's ships wins.





Implementation

Step 1
When the program runs, it will prompt the user to enter a filename if one has not already been provided as a command line argument. The file will be a .txt file that holds a battleship map, and the map's associated high-scores.

A sample is shown here:

Highscores:
1. Player1
2. Player2
3. Player3
4. Player4
5. Player5
6. Player6
7.
8.
9.
10.
XXXXXBBBBX
XXXXXXXXXX
XXAAAAAXXX
XXXXXXXXXX
XXXXXDXXXX
XXXXXDXXXX
XXCXXXXXXX
XXCXXXXXXX
XXCXDDXXXX
XXXXXXXXXX

Note: This sample file will be used in the samples displayed in this document.

There will always be 10 player ranking positions – even if fewer than 10 people have played the game. Also, the game board will always be a 10x10 grid of characters. It is possible for users to accidentally type in a text file that doesn't exist – or to have a text file that is not formatted properly.

Step 2
Now, make sure to check if the file contains valid ships.

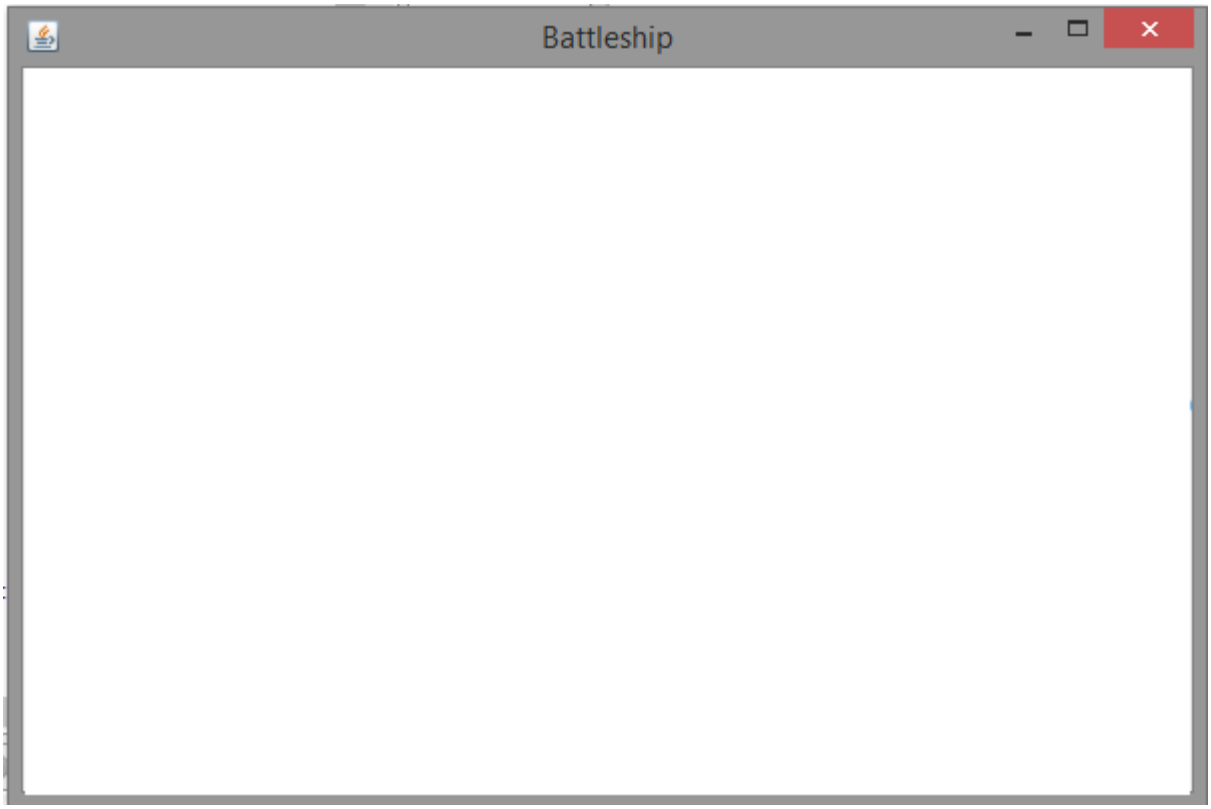
Ship Name	Number of Ship	Size of Ship	Ship Marker
Aircraft Carrier	1	5	'A'
Battleship	1	4	'B'
Cruiser	1	3	'C'
Destroyer	2	2	'D'

Note: There are many ways to do this – some are exhausting. If you find yourself spending too much

time on this, brainstorm for a bit.

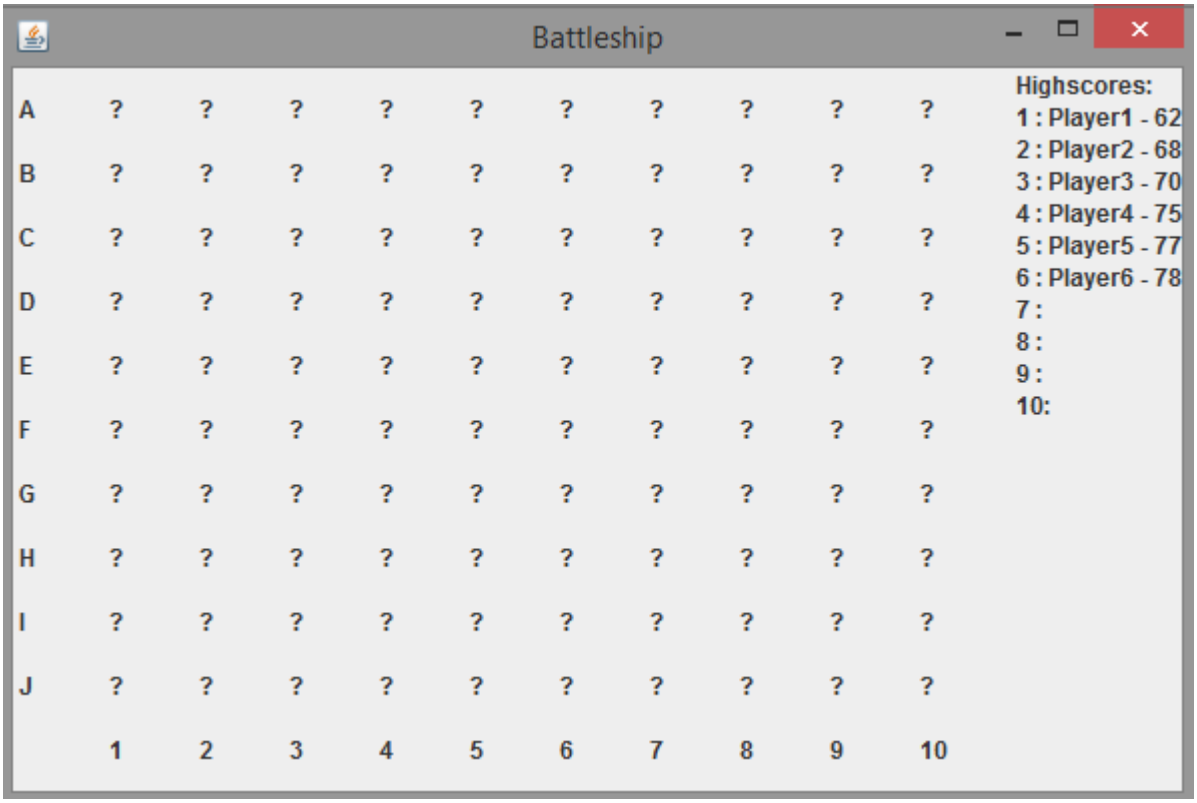
You will need to save the information about the ships and the coordinates it spans somehow. Think of good object-oriented principles here.

Step 3
Now that we have all the information needed to create a battleship game, go ahead and create a simple window entitled “Battleship”.



The sample shown above is 600x400. You may, however, deviate from this. Just make sure that the title is “Battleship” (Or something similar – don’t just leave it blank).
The window closes when the ‘X’ is clicked.

Step 4
It’s now time to add the elements of our game. We want the user to see a grid along with the high scores. The layouts you use are up to you, but the result should look similar to this:



Step 5
The game looks great! There's only one massive issue – the user can't play the game. Go ahead and add the following prompt to display in the command line:

```
Turn 1 - Please enter a Coordinate:
```

Or something similar. The user will type coordinates to guess – make sure that they can only enter values A-J and 1-10. The user will format the coordinates like so:
B6
H9
A10
C7

Once they enter a valid coordinate – one of three things will happen:
The user's guess is a miss.
The user's guess is a hit.
The user's guess was already guessed.

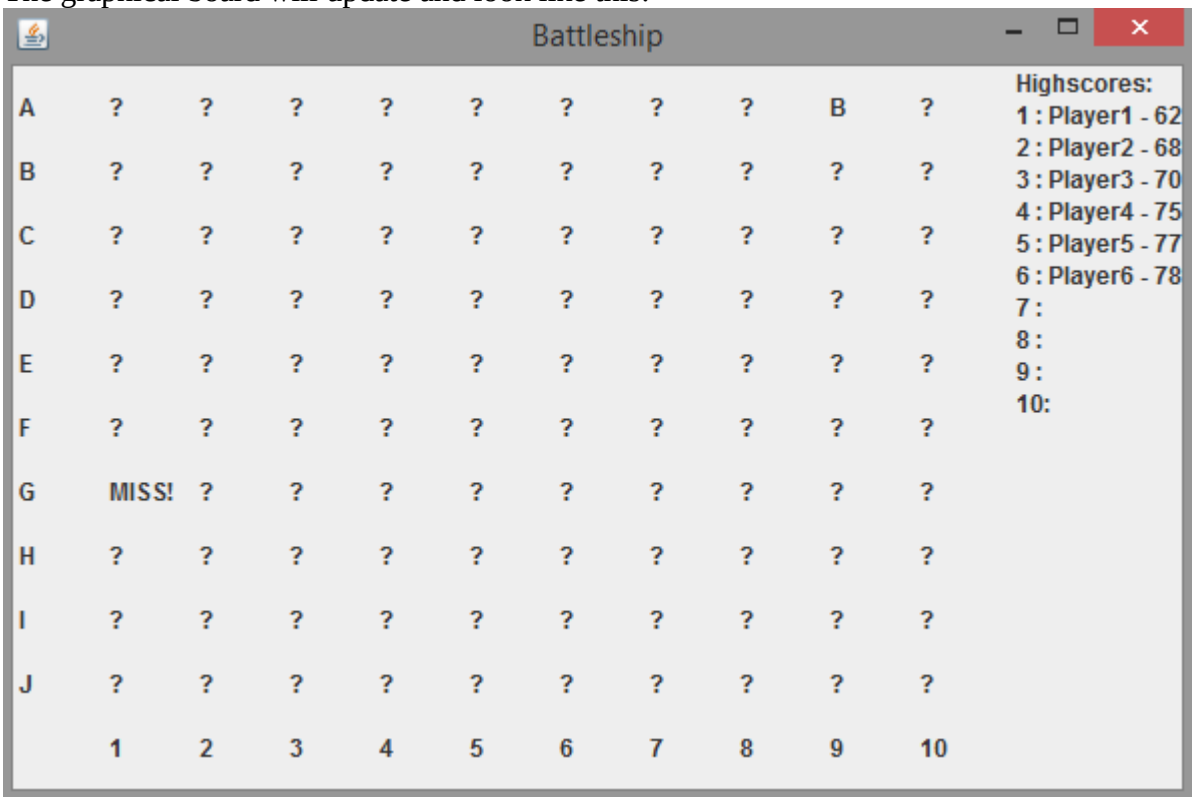
If the coordinate was already guessed, the Turn will not increase and will re-prompt the user to try again.

Otherwise, the Turn count will increase and update the game board. If it is a hit, the space will display the ship's marker; if it is a miss, the space will display 'MISS'.
Hint: Call the .setText(String) method on a JLabel to update what it displays – you don't have to make a new JLabel.

Here is a sample run:

```
Turn 1 - Please enter a Coordinate: A9
You hit a Battleship!
Turn 2 - Please enter a Coordinate: Q32
That's an invalid Coordinate, try again!
Turn 2 - Please enter a Coordinate: G1
You missed!
Turn 3 - Please enter a Coordinate:
```

The graphical board will update and look like this:



Step 6

Now, the user needs to be able to sink a ship. Simply output a message of the type of ship sunk once the user guesses all of the locations of a certain ship.

Here is a continuation of the sample run, hitting A8,A7, and A6 – sinking the battleship:

```
…
Turn 5 - Please enter a Coordinate:A6
You hit a Battleship!
You have sunken a battleship!
Turn 6 - Please enter a Coordinate:
```

The graphical board will update as normal – nothing special happens graphically when a ship is sunk.

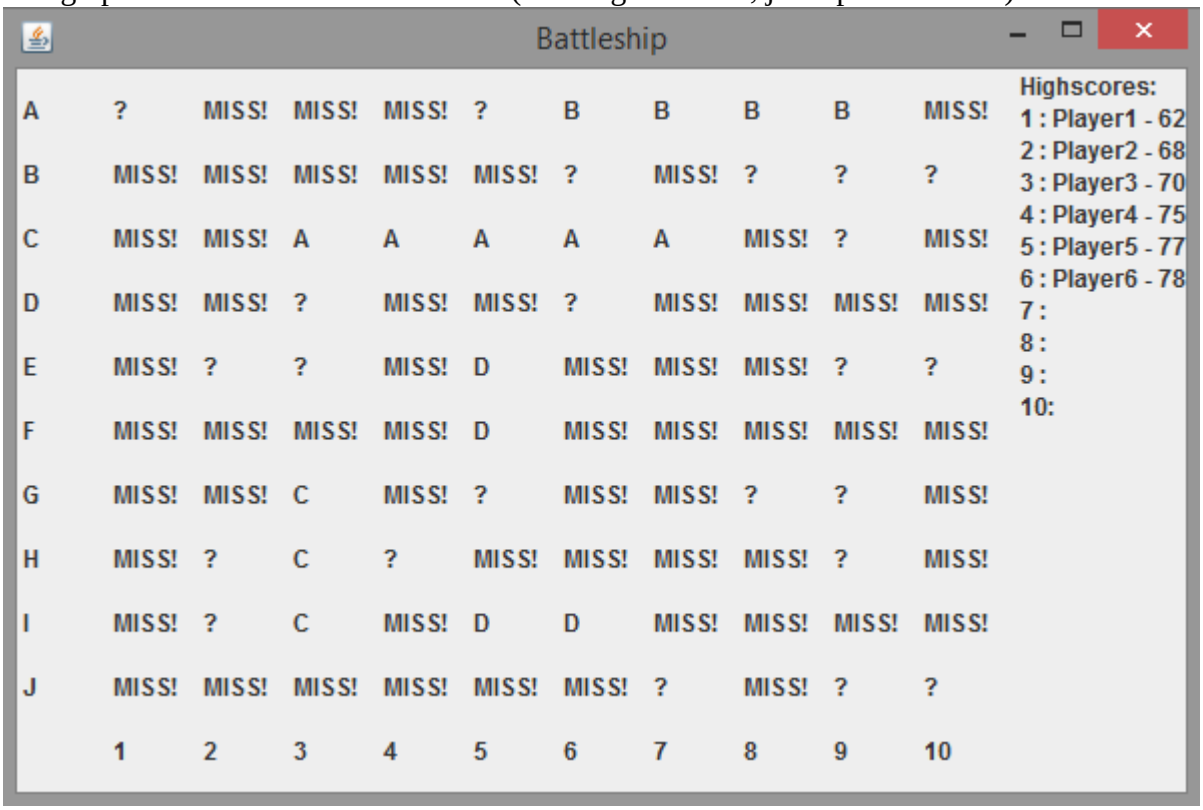
Step 7

Now that the user can sink ships, the win condition – sinking all the ships – can occur in the game. When the player wins, they will enter their name. The high-score table will then be updated if the user beat any of the other players.

The window will swap to a screen showing the updated scores and a ‘Thanks for playing’ message along with their entered name and score. Their score is simply the number of turns they took to beat the game.
Note: Keep in mind, a high score in battleship is actually the lowest score.

Here is a continuation of the sample run, scoring a near perfect score of 77 (Ah well, better luck next time).

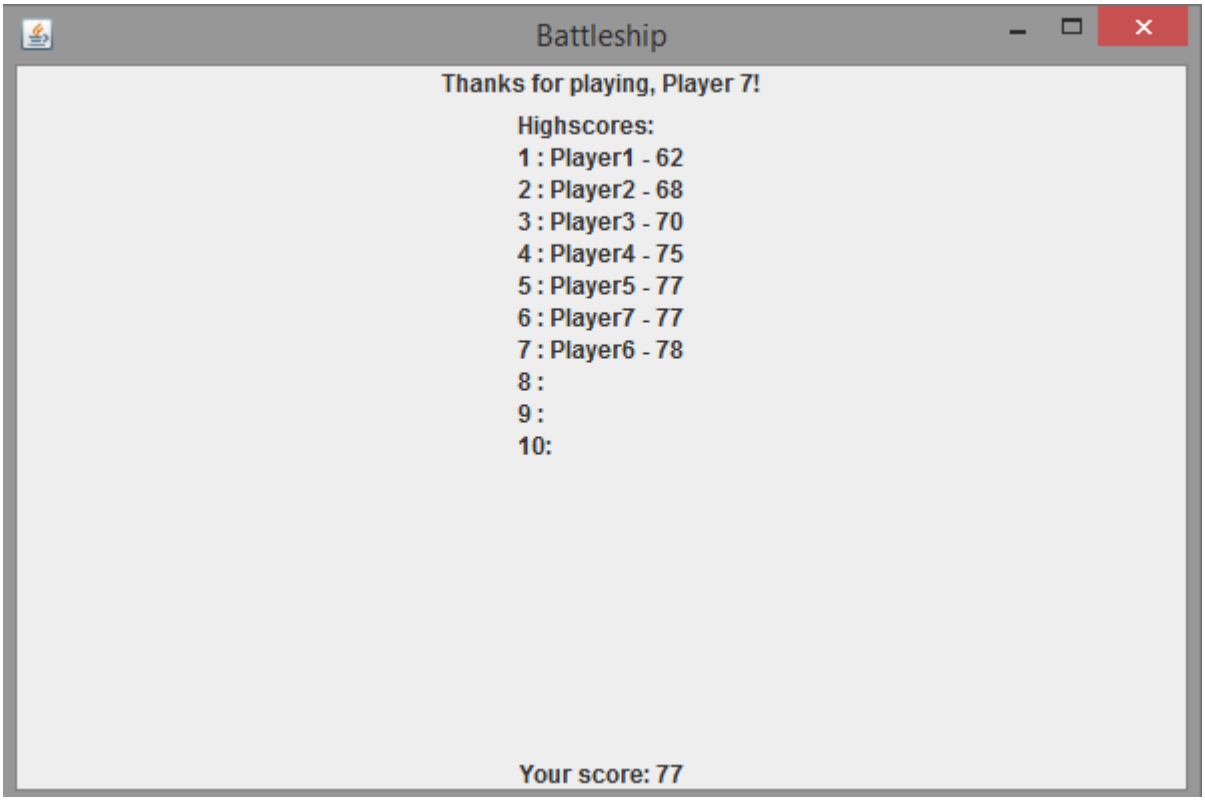
The graphical board will look like this: (Nothing new here, just updated labels).



The user then enters their name:

```
…
Turn 77 - Please enter a Coordinate:F5
You hit a Destroyer!
You have sunken a Destroyer!
You sank all the ships!
Please enter your name:Player 7
```

The graphical board will go away, and be replaced by the screen described earlier:



You can also see that the player came in 6th place. Although Player7 tied with Player5, they should not get ranked above them.

Note: If more than 10 people play, the scores worse than 10th place will not be recorded – only the top 10.

Make sure to save the new leaderboard to the file from which it was read.

Grading:
This assignment will be graded by using a series of inputs to test your program. We will use a variety of good and bad input to test. A general rubric has been provided below, but if your program gets marked down, the category and the point values will be listed on Blackboard. We will release the exact test cases used after the assignment is due. That way, you can test it yourself and check the grade given to you. Partial credit is given.

Category and % of Grade	Criteria
File I/O 0.75%	
0.2% -	Reads in high-scores.
0.2% -	Reads in battleship map.
0.35% -	Outputs correct update to file when a high-score is made.
GUI 1.0%	
0.1% -	Window is set-up as described.
0.2% -	Grid is set-up with letters/numbers on the sides.
0.2% -	A High scores list is displayed to the right of the game board.
0.2% -	The end screen displays the message, scoreboard, and individual score.
0.3% -	Layout/formatting resembles the samples shown.*
Game 0.75%	
0.1% -	The game only accepts valid input.
0.35% -	The game responds correctly to ships being hit/sunk.
0.4% -	The game board updates correctly on each valid move.

*You may not use null layout. The grading for this won't be too strict – however, things should look neat and be in the same relative locations as in the sample.