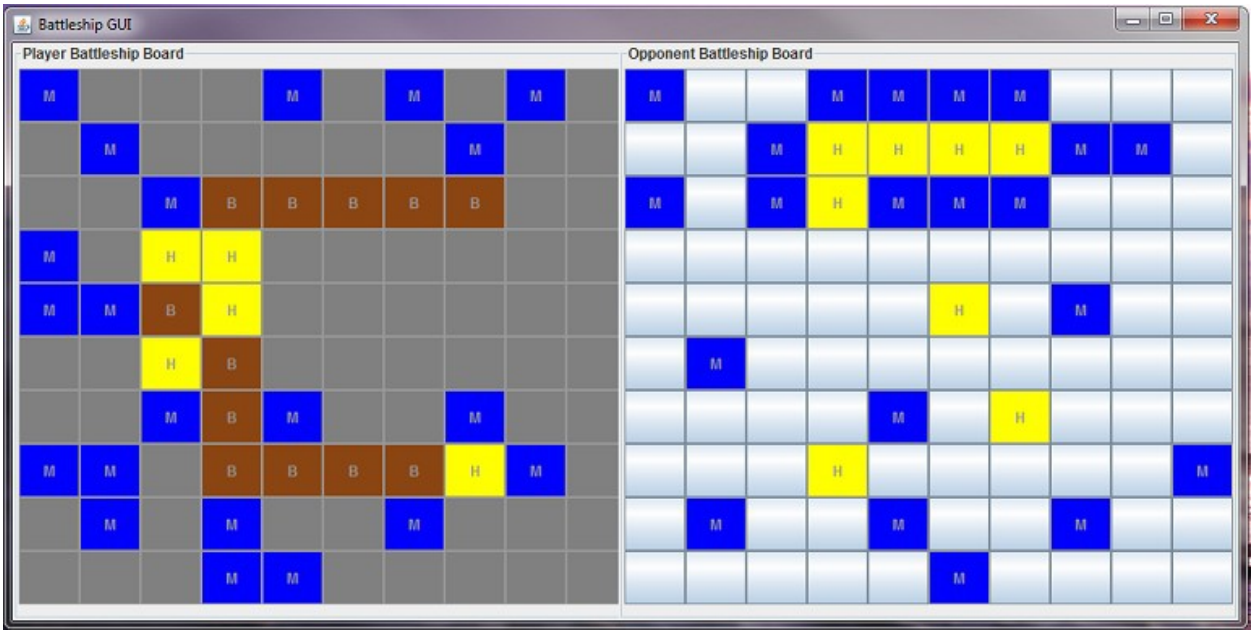
CS162 - Assignment 2: The Full Battleship Game

This assignment is intended to give you experience with designing an object-oriented program with multiple classes. You will also get practice with Graphical User Interfaces, Java Interfaces, Inheritance, and Polymorphism.



Assignment Overview:

You will be implementing a simplifed version of the full Battleship game. In this simplified version, you do not need to worry about interactively placing the battleships on the board. Instead, you will place the ships by editing a text file, and then loading the text file for your board. To take a look at what this text file looks like, take a look here. The file represents a 10 x 10 board, with the '.' character representing an empty space. The ships are numbered 1 through 5, and each occupied square in the text file is filled in with the corresponding number for that ship. The numbers, along with the sizes of the battleships are as follows:

- (Index 4) Aircraft carrier (Size 5)
- (Index 3) Battleship (Size 4)
- (Index 2) Destroyer (Size 3)
- (Index 1) Submarine (Size 3)
- (Index 0) Patrol boat (Size 2)

The opponent's board is stored in a text file with a similar representation. You do not need to write code to handle the loading of the board. We have provided a BattleshipBoard constructor that takes a filename as an argument. Use this constructor to load the board. When you run the game, you will need to specify the file names for the player's board and the opponent's board as a command line argument.

Your implementation must then play battleship according to the standard turn based rules of the game. If you are not familiar with the rules (Milton Bradley version), look at this Wikipedia article. Basically, you and your opponent take turns firing on each other. The first player to sink all of his or her opponent's ships wins.

Requirement 1: Your game should display two boards (see the screenshot above). Your display does not need to match the screenshot exactly, but it should have the same functionality:

1. The first board contains the locations of your 5 battleships and it also records the shots taken by your opponent. Your GUI must update this board every time your opponent takes a turn.
2. The second board allows you to specify the locations of your shots. You will shoot by clicking on an appropriate square. This board also keeps track if the shots you took were hits or misses. You need to display hits and misses differently (you can change the color, change the symbol on the cell, etc. -- how you actually do this will be left up to you).

Requirement 2: You must also implement a computer player. The computer player needs a strategy for deciding what shots to take. For this assignment, you must implement two different types of strategies (below). **You must use either inheritance or interfaces for these two types of strategies**.

1. **The systematic player**: The systematic player starts at (0,0) and systematically shoots one square over from its previous shot. If the next shot goes beyond the edge of the board, the systematic player starts again at column 0 on the next row. As an example, the systematic player will shoot at (0,0), (0,1), to (0,9) and then start over at (1,0). You may create a smarter version of the systematic player that shoots two squares over and alternates starting at column 0 and column 1 on the next row.
2. **The random player**: The random player randomly selects a position on the board and shoots at it. If the player has already shot at that location, the player picks another location and shoots at it. To select random numbers in Java, use the Random class that is part of Java and use the nextInt() method.

Before your program starts up, you will need to specify what type of computer player (ie. systematic or random) you want to play against. This will be done as a command line argument (see below)

Requirement 3: Finally, you need to indicate who won and that the game is over. You can do this any way you like (eg. a JLabel or a dialog box) but you need to notify the user.

Before you begin, it is a good idea to design your game using the principles of Object-Oriented design that we discussed in class. I recommend planning this out on paper first. Keep in mind that the overall design of your code will be graded. You can ask a TA or the instructor if your design is a good one.

Command line arguments

If you don't know what command line arguments are, take a look at these slides. Write a **main** method, which will accept three command line arguments

- **Parameter 1:** Path to the playerBoard File (See playerBoard.txt in the "Files You will need: " section).
- **Parameter 2:** Path to the opponentBoard File (See opponentBoard.txt in the "Files You will need: " section).

- **Parameter 3:** A character (either 'r' or 's') representing which type of computer player to use.

Here is an example of a command line (inside the "Program arguments" box in Eclipse):
opponentBoard.txt playerBoard.txt r

Bonus

The bonus for this assignment will be a free-form bonus. You can try adding anything that you think will impress your graders. You may earn up to 10 bonus marks, with more difficult additions earning you more bonus points. **If you attempt the bonus, remember to hand in a text file called "bonus.txt" describing what you added**. Examples of things you can add include:

- Make the display nicer by adding color, etc.
- Add sounds effects. If you do this, please remember to hand in the sound files.
- Come up with a smarter computer player
- Allow interactive placement of the battleships by the human player
- Come up with a way to automatically place the battleships for the computer player

Files

We will provide the BattleshipBoard class to help you. You can use it if you like. You may modify any of the code we provide, but your code must be able to load the player and opponent board files using the same format as the ones we provide. Files you will need:

- BattleshipBoard.java (Use this BattleshipBoard instead of Assignment 1's BattleshipBoard)
- **BattleshipException.java** (The Exception class used by the Battleship game)
- **Cell.java** (This is a very simple class that stores the row and column)
- **FireButton.java** (This class extends the JButton class with a Cell object. This allows you to return the row and column if you click on a Cell)
- **playerBoard.txt** (A file representation of an player board)
- **opponentBoard.txt** (A file representation of an opponent board)

Hints

- The components you need for a basic version of the game: a JFrame for the entire game, 2 JPanels with GridLayouts, and lots of FireButtons.
- The main function in BattleshipBoard may be confusing some students. The main method should really either be: a) in a class by itself or b) in a top-level class that runs the whole game.
- In your ActionListener's actionPerformed method, you will notice that it takes an ActionEvent object. If you registered the ActionListener with a FireButton, you can get the FireButton object that you clicked on by doing the following:

```
public void actionPerformed(ActionEvent arg0) {
   FireButton fb = (FireButton) arg0.getSource();
   // more stuff here...
}
```

Live Example (Applet Version):

Click HERE to view a live example of BattleshipBoardGUI.

- Select which type of BattleshipBoardGUI you want to see, by selecting the appropriate button.

What to Turn In: (Remember to turn in the .java files NOT the .class files)

Please hand in all the .java files for your assignment. Turn them in via Blackboard and TEACH.

- All .java files for your assignment. Do NOT hand in the .class files. If you have more than 10 Java files, please zip up your handin files and submit as a .zip file.
- If you attempt the bonus, hand in a file called bonus.txt that describes what you added.

Grading Scheme:

You will be graded on the functionality of the game, the use of correct coding conventions, and the overall design of your project.