CSCI 201L Assignment #3
Spring 2015
4.0% of course grade

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

Overview

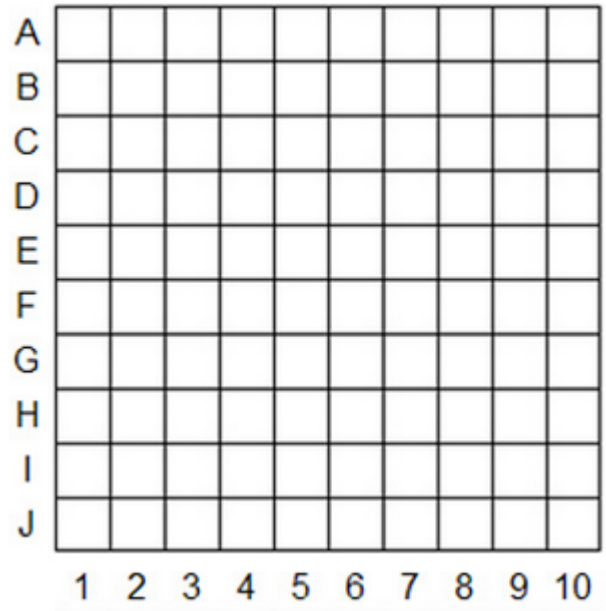
This assignment will build off your previous assignment to improve your Battleship game. Luckily, all of the major components are already done from the last assignment. You will need to tweak some of the ways the program does things and add some better graphics to give a more polished look. You will still be creating a 1-player battleship game, but now a computer will play against you. You will have two Battleship maps – one representing your ship layout that the computer guesses and the other representing the computer's layout that you guess. You will be removing the high score table.

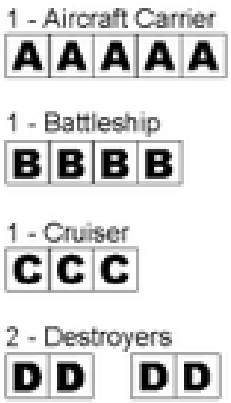
Introduction

If you are worried that your code isn't up to par to continue, don't worry. We have provided you with a solution to the previous assignment in order to complete this assignment. We highly recommend that you use your own code though – it is an important skill to be able to build off old code and maintain good reusable code. In addition to this, it will be easier since you should already be familiar with the code you built.

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.
NOTE: This is the same background as last assignment – this is for easy reference.



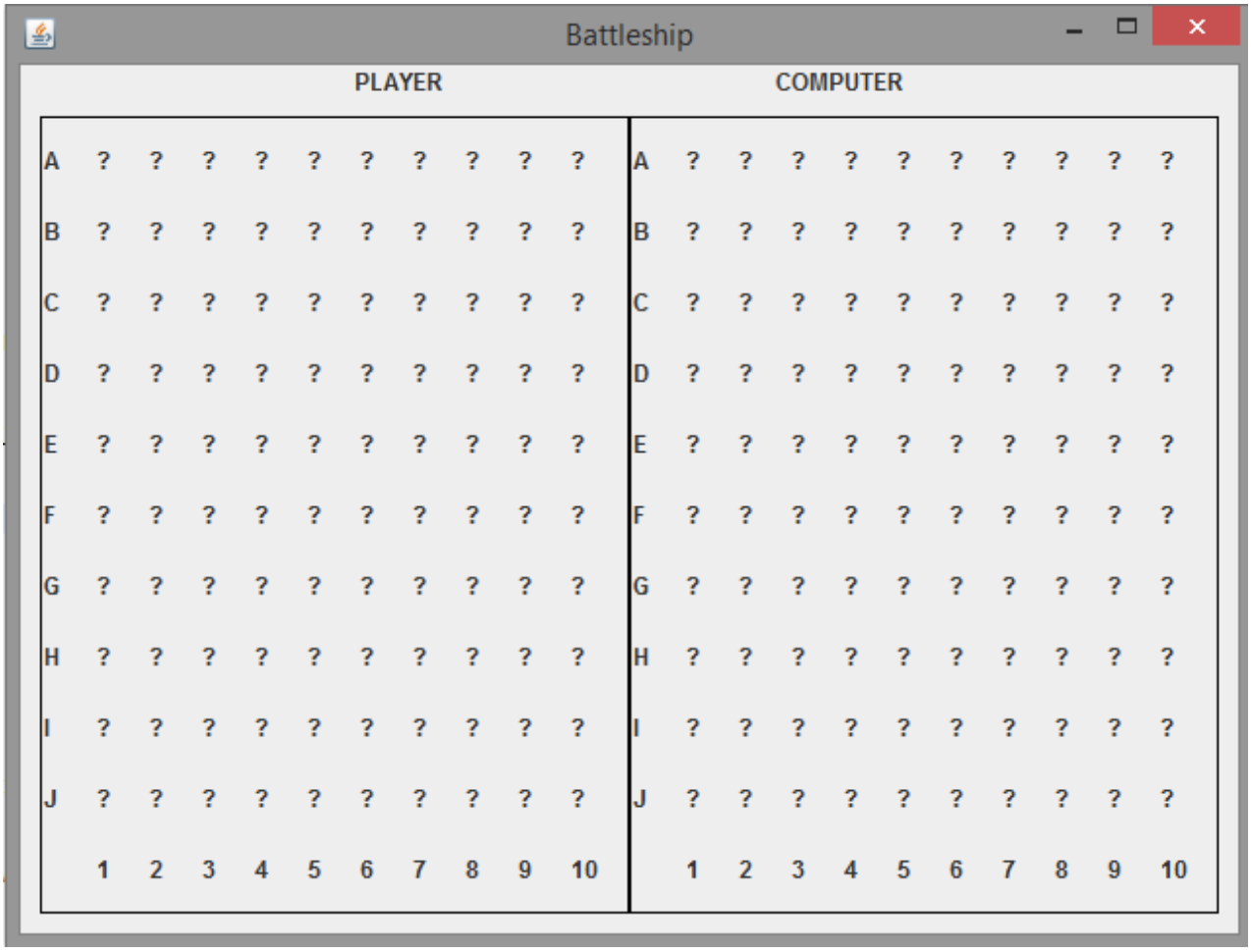


Implementation

Step 1

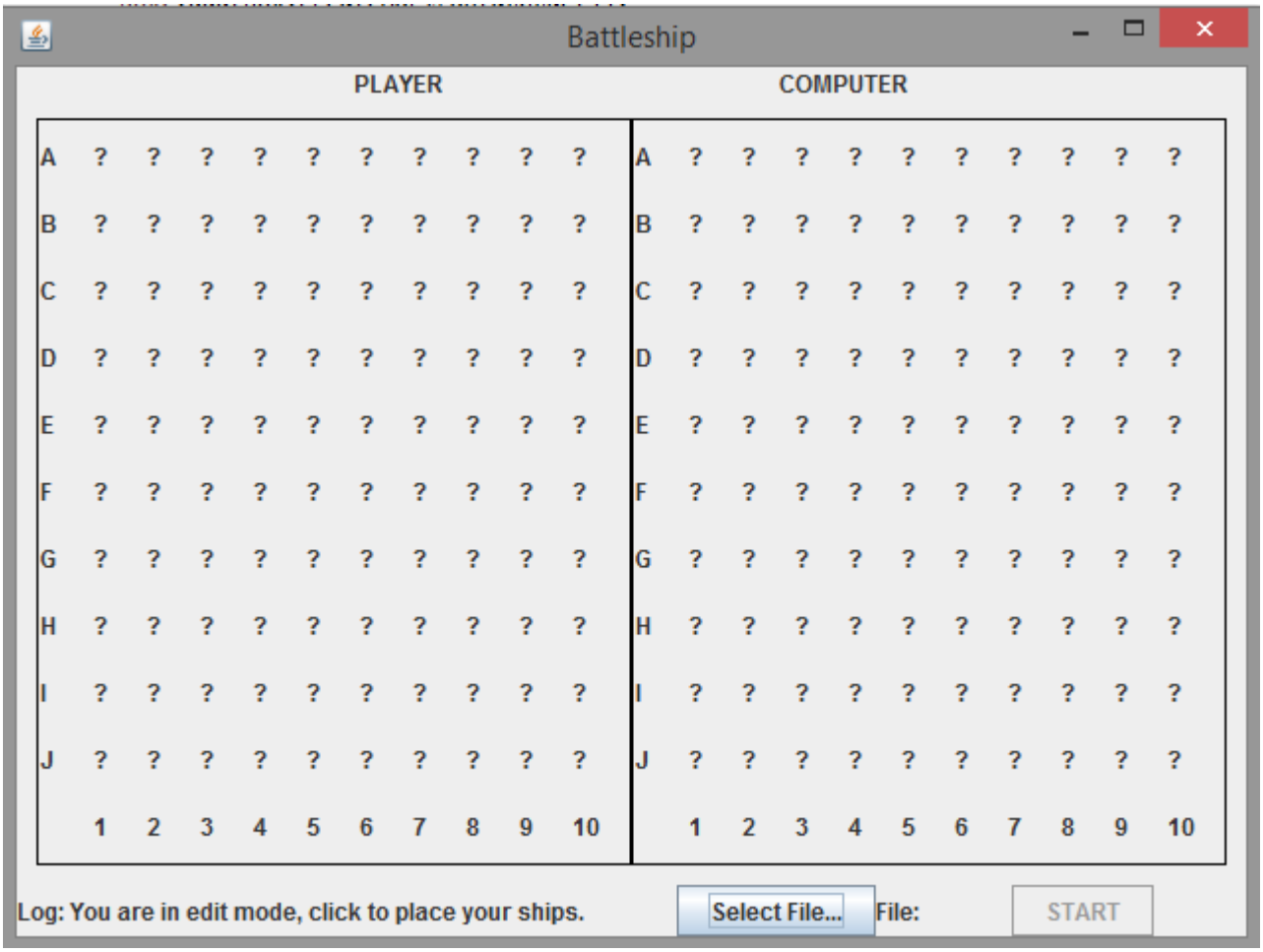
Take a step back and look at our battleship program. First, we don't need the high-score table anymore. Since the player is playing against a computer, and not against other players' scores, it is just eating up real-estate space, so we will remove it from this assignment.

Now we need to add a space for the second grid. We should be able to reuse most of the code that we made to generate our main grid from the previous assignment. It may look cluttered, so add some dividers between the grids. Also, add labels so we know who the player is and who the computer is.



Step 2

We must let the player choose his coordinates. In addition to this, we must have the computer initialize its own coordinates. We can do this by making an "edit" mode. The program will open up to this mode and only return to it once the game is over. Go ahead and add the following to your GUI: A "Select File…" button, a "Log:" JLabel, and a "START" button.

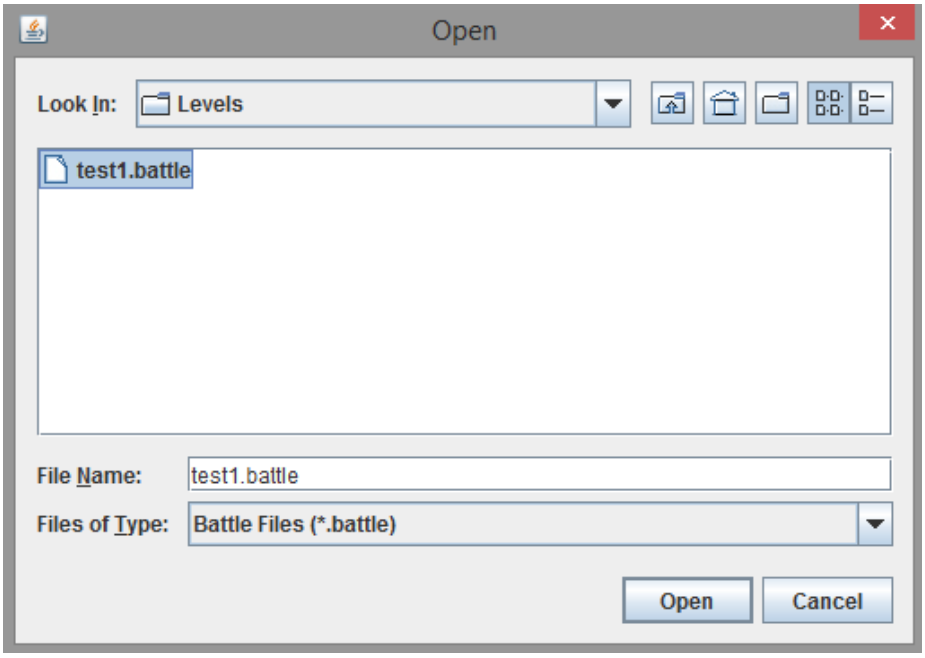


You will be using the Log for the entire game – this is additional information about what is currently happening in the game. When the program starts in edit mode, it will inform the user that they are in edit mode.

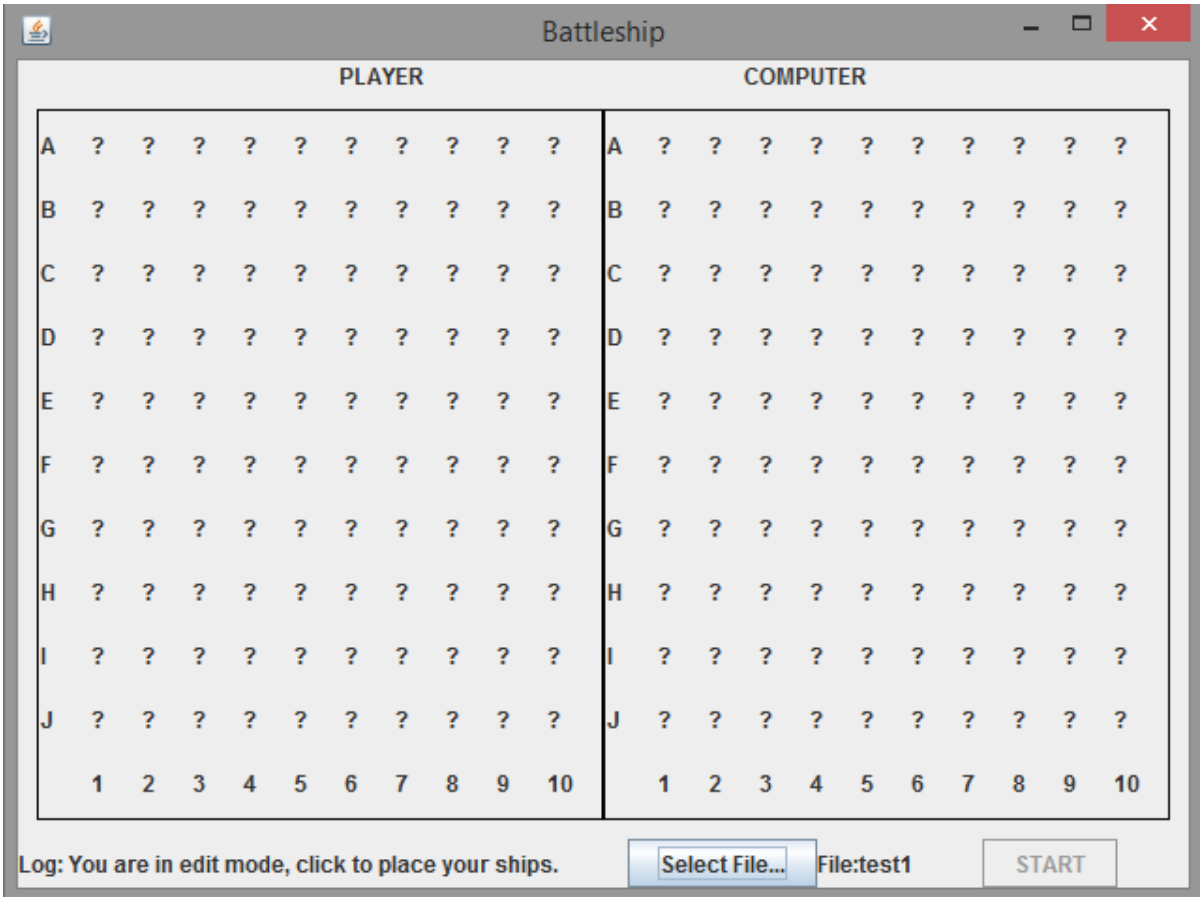
The Select File button will open up a JFileChooser. The user must select a .battle file, which will contain the coordinates of the enemy ships.

"A .battle file? I've never heard of that before!?"
Exactly. We can create a new type of file simply by adding a unique extension to it. This lets programs know that the file's information will match what the program is looking for. There is nothing "special" or "magical" about different file extensions – they just let a program know what to expect. This way, we can only accept .battle files, and we know (hopefully) they will work with our program whereas a .txt file could have been a file unrelated to our game. They will follow the same format as in the previous assignment, but without the information about the high scores.

"What about an invalid .battle file?"
Later on, our program will generate its own .battle files from our map editor. The only way bad .battle files will be created is if a user edits them manually – that risk is on them. For now, we will assume that all files will be valid. We won't test invalid files for this assignment.

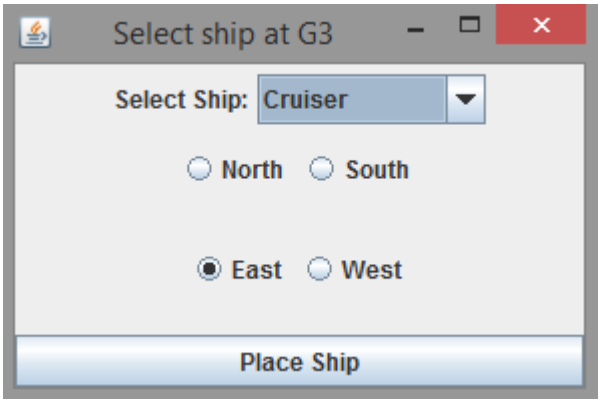


Once a .battle file is selected, append the name of the file to the JLabel “File:” without the extension.



NOTE: The labels should not change after selecting a file, the ships should be unknown to the player.

Step 3
Now we need the user to be able to place their own ships. Do this by opening a pop-up window when they click on one of the '?' labels on their side of the board. The window with a title indicating the head of the ship will show a list of what ships can be placed and offer an orientation. If the placement is invalid, the "Place Ship" button must be disabled. The user must select a valid ship and orientation to enable the button, then they may press "Place Ship". The letters of the ship will appear on their game board.



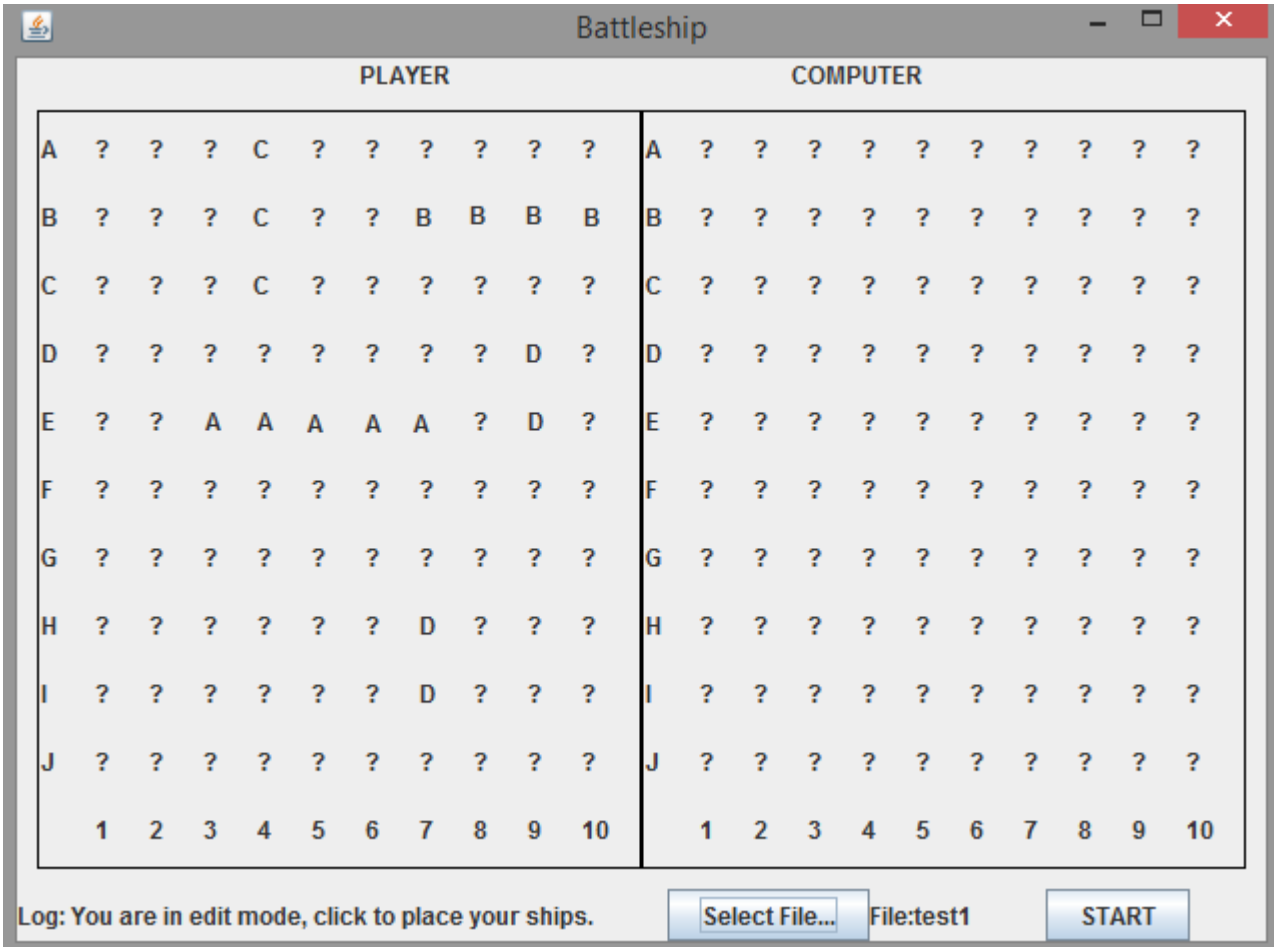
The orientation of a ship denotes which way the ship is facing. If the above was chosen, a Cruiser would be placed at G3,G4,G5.

If a ship were to go out of bounds or cross another ship, the selection is invalid. If a ship is placed, the next time the user tries to place a ship, that ship will not be provided as an option (unless it is a Destroyer, then the selection must be made twice in order for it to no longer appear).

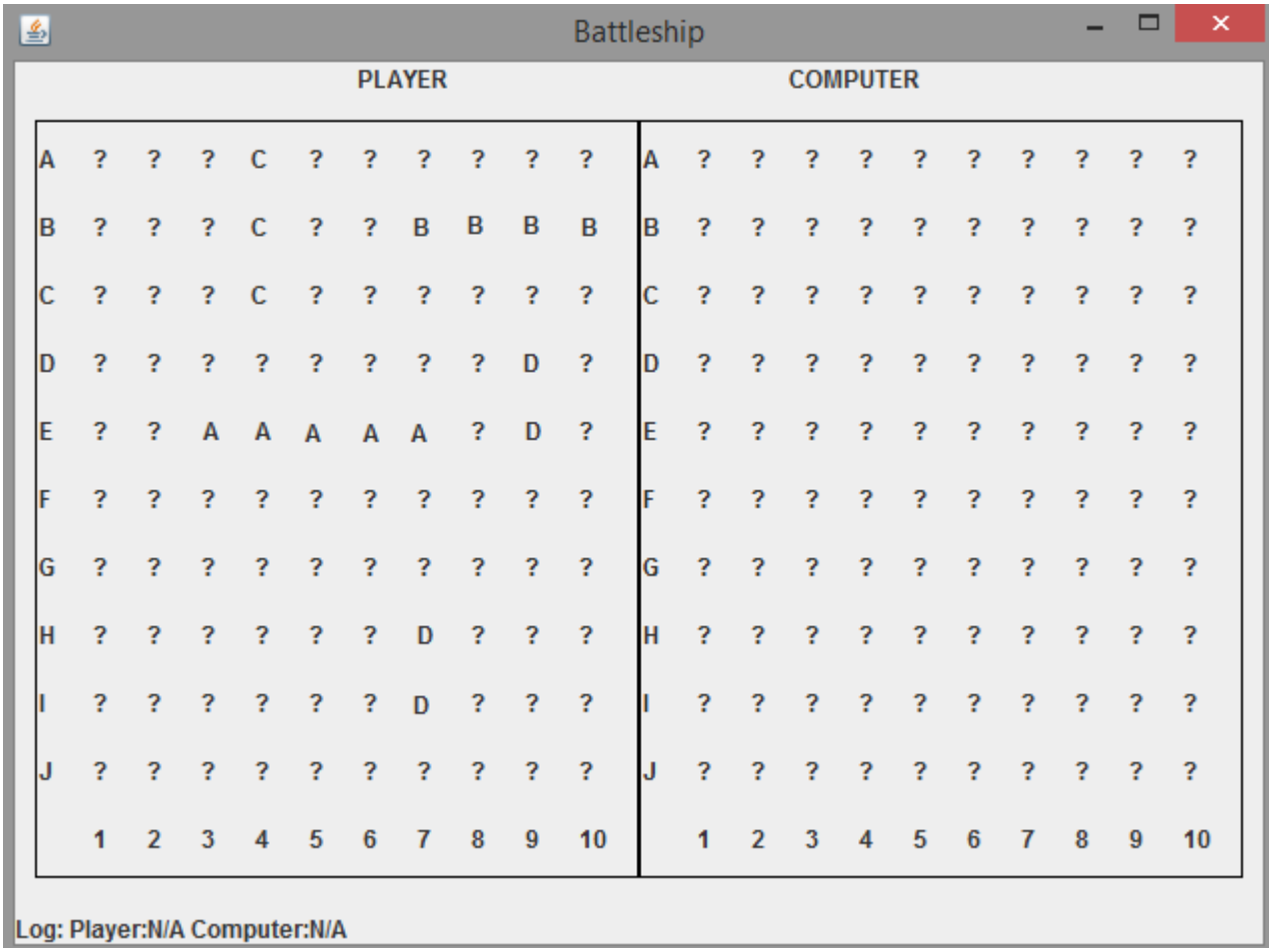
To clear a selection, the user must simply click on any part of the ship in edit mode.

Once all 5 ships have been placed, this pop-up menu won't appear when a ? space is pressed.

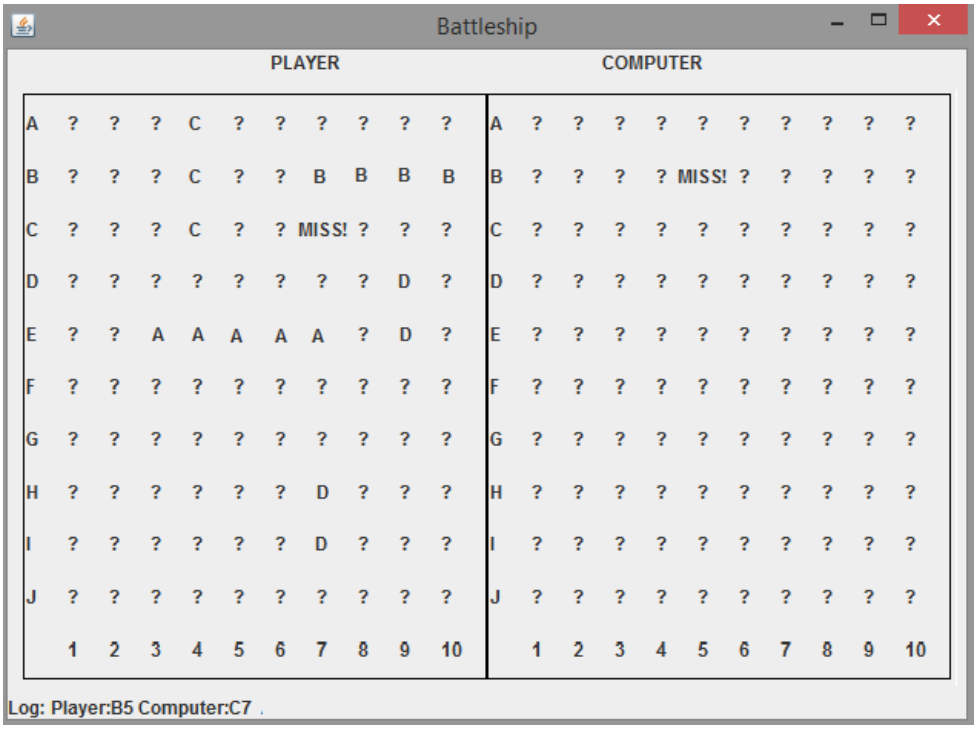
Step 4
Now that the user can enter in their desired ship positions and select a .battle file for the computer to use, we are ready for the user to be able to play the game. Once all 5 ships are placed, and a .battle file is selected, the "START" button should be enabled.



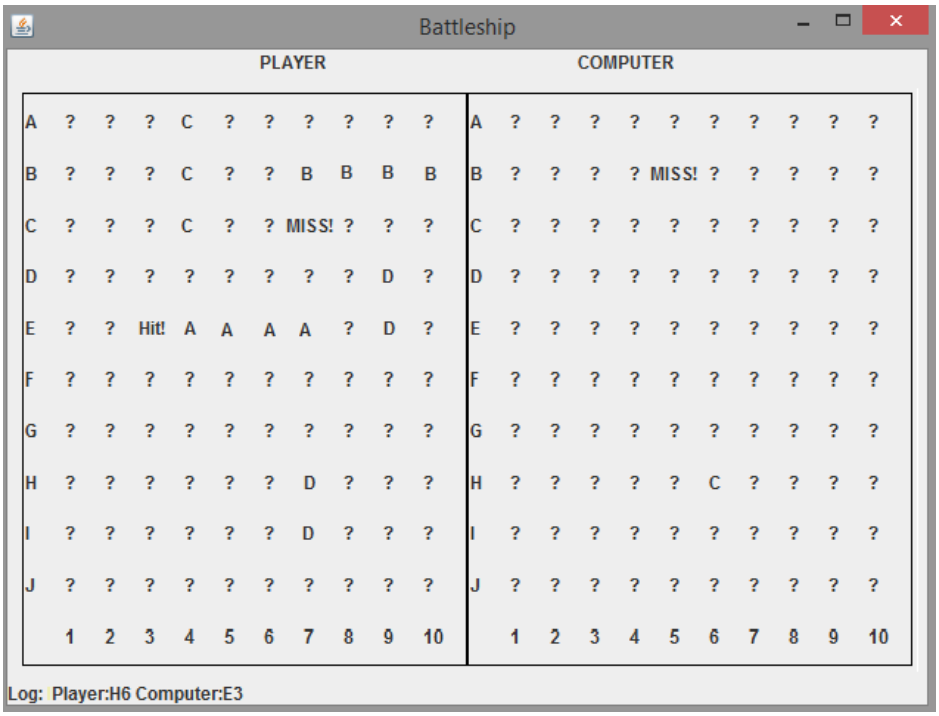
Once the user presses start, no editing should be able to take place. Clicking on their side of the board should not do anything.
The Log will update to “Player:N/A Computer:N/A”
The “Select File…” button and the “START” button will disappear.
The boards will act exactly as they did in the previous assignment, however this time the computer’s board will update via click instead of from console input. The player’s board will update exactly when the computer’s updates – but the selection will be randomly generated by the computer.



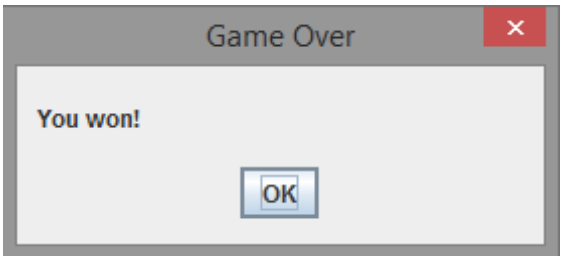
The N/A stands for Not Applicable. This is here because it is the first turn. Once a player selects a valid spot, the computer will also select a valid spot. The boards will update, and the Log will display where the player hit, and where the computer hit.



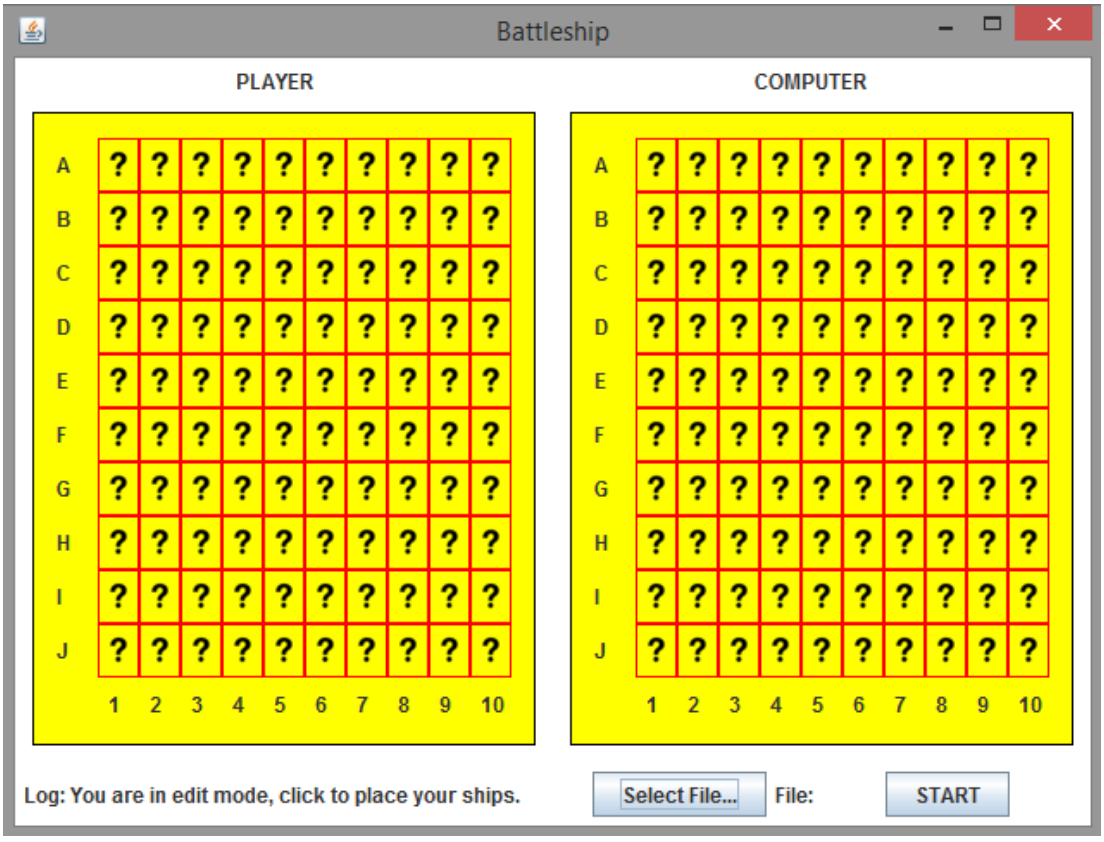
In the example above, both the computer and the player missed. If the player hits a computer's ship, the label would be the corresponding letter. What happens if the computer hits the player? The label is already corresponding to the letter of the ship! We need to make sure if a player's ship is hit, a different label is displayed. Display the text "Hit!" in the space if the computer hits a player's ship, replacing the letter corresponding to the type of ship. In the following example, the player successfully hit part of the computer's cruiser ship. This is represented exactly how it was in the previous assignment. When the computer hit part of the player's aircraft carrier, a 'Hit!' mark is put in place of the 'A'.



Step 5
The game will end once the computer or the player guess all the ship coordinates of the opposing player. When this happens, a dialogue window will appear letting the user know if they won or lost. The user won't be able to interact with the main window. Once the user exits the window, the game will reset and go back into edit mode.

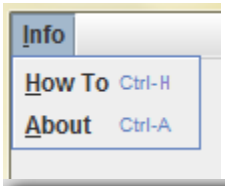


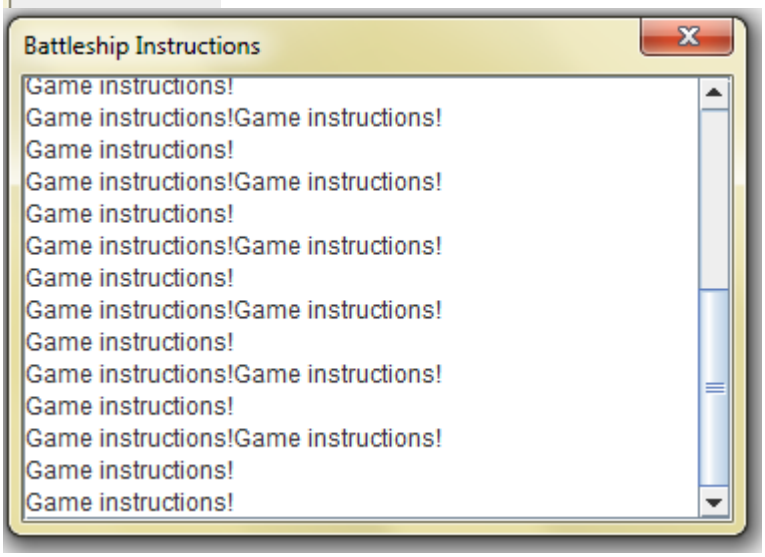
Step 6
Now that everything works, we can make the Battleship game look a lot better. It's pretty boring to have plain letters and numbers as the grid. Sample images that correspond to the game are provided for you. However, feel free to make or use your own. This should be a surprisingly simple transition – JLabels can display images as well as text. Simply map the strings you used for your current JLabels to images. This will hopefully make our game look a bit more professional. In addition to this, adjust and clean up your window. Make sure nothing is crammed into the edges, and perhaps add some color. Just 5-10 minutes of time can really polish the finished product.
NOTE: Borders are used on the JLabels to form a grid.

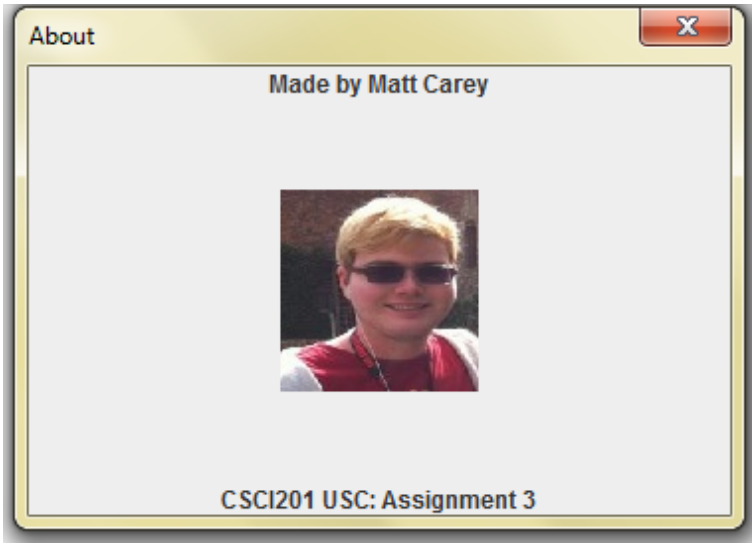


Step 7

Lastly, we need to add an Info menu. One option will be “About” – this will show some information about the program - who made it (hopefully you), the date it was made, and the assignment number. The other option will be “How To”, which will show a scrollable text area where instructions on basic gameplay will be displayed. It is a good idea to store this information in a text file. That way, when you build upon the game, you can just update the .txt file manually. Make sure to add accelerators for quick access.







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. 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 graded so you can test it yourself and check the grade given to you. Partial credit is given.

Category and % of Grade	**Criteria**
Updated GUI 1.0%	
0.2%	Two grids that are each labeled “Player” or ”Computer”.
0.2%	A Log displays current information.
0.2%	The “Select File” and “Start” button appear in editor only.
0.2%	“Select File” button opens a file chooser that only selects .battle files.
0.2%	“Start” button is only active when the game is ready to be played.
Game Editor 1.0%	
0.2%	When a cell is clicked and is empty, an edit window appears.
0.2%	When a cell is clicked and is occupied, the associated ship clears.
0.2%	The pop-up edit window is formatted and works properly.
0.1%	Only valid ships appear from the dropdown menu.
0.1%	Only one direction button can be selected at a time.
0.2%	Only valid selections are applied to the game board.
Gameplay 1.0%	
0.2%	Game log updates correctly.
0.4%	The player uses mouse clicks to make a move.
0.3%	The computer makes a random move when a player moves.
0.1%	On win or loss, the window goes back to a cleared edit mode.
Graphics 1.0%	
0.2%	The board uses images instead of Strings to represent the game board.
0.2%	The UI is improved significantly for a polished look.*
0.2%	The menu uses accelerators to open two pop-ups.
0.2%	The “How To” Menu is as described.
0.2%	The “About” Menu is as described.

*You may not use null layout. The grading for this won’t be too strict – however, follow the sample. Feel free to get creative.