Battleship Game

Project Documentation

Team: Exhalation

Team Member Names: Jonathan Bluhm, Ethan Bilek, Ziad Alwazzan, Yosan Russom

Install

Download file “battleship.jar” from https://github.com/johnnybluhm/OOP-project

Go to your console/terminal and navigate to the directory containing the file

In that directory type "java -jar battleship.jar"

If you don’t have java installed, go here:
https://java.com/en/download/help/download_options.html

Brief description of project

The project is a modified version of the classic game Battleship, using the coding language Java to create it. The player will play virtually against a computer controlled opponent. All of the classic rules apply - you have the ability to place your ships down on a grid and the ability to attack your opponent’s grid. The goal of the game is the same as well, to sink all of your opponent’s ships before they sink yours! There are also some additional features that were developed as time went by which are explained further under “User Stories” below. Players have two special power ups: an airstrike and a sonar pulse. The players are also able to move their ships after they have been placed. In case a player places a ship down in the wrong place, moves their fleet, attacks a point on the grid, or uses their airstrike somewhere they did not mean to, there is a feature that allows them to undo their actions (and redo them if they want to go back to how things were). Storms will also randomly appear on the board and can sink a player’s ship if big enough.

It is also worth noting that this project was specifically made for the course CSCI 4448/5448 at CU Boulder in the Spring semester of the 2020/2021 school year in order to learn Object-Oriented Design. The goal of the project was to create a working game of Battleship, yes, but also to find a way to incorporate different coding patterns (taken from the book *Head First Design Patterns*, second edition) into a simple coding base as well as learn how to refactor code.

Development Process

We did a modified version of extreme programming. Each week we would meet and discuss what we needed to do, and then begin to work on what we could do together/on our own over the following days. For our coding process, we would write the tests for our code and then program the feature. We focused on getting our tests to pass so that when we wrote new features we could easily tell what worked and what didn't. Once our tests passed we would go back and refactor to make the code more readable by humans.

On top of this, we were often given new features that we needed to implement every other week. Initially we just had the core background of the game of Battleship, and as the weeks went by we needed to add features like the sonar pulse and airstrike, amongst other things. This is where the extreme programming method was nice. We would write the tests for the features so we knew how we wanted them to work, then we would write the code so they would pass the tests, then we would refactor our code not only to make it more readable as mentioned above but also to make the new features work well with the old ones.

Unfortunately, when it came to the graphical user interface (GUI), standard methods were abandoned. Instead of using functional tests, we opted for simply changing some code, and verifying that the GUI was looking how we wanted. Eventually we found that the look and feel of the GUI was adequate, and there was no need for further testing.

User Stories

Battleship game

Jeff is a 10 year old boy who loves running around outside.
Jeff opens up battleship and is immediately transported into a world of war. He is prompted to select the difficulty of his opponent. Three options appear: easy, hard, and you're gonna lose. Jeff, being the brave and foolish child that he is, selects you're gonna lose. He is then prompted to place his 3 ships. Jeff is amazed at how he gets to choose where to hide his ships. He places them, and then begins attacking his opponent. Jeff shouts for joy when he realizes he got a hit on his opponent in his first try. He knows he hit an opponent's ship because he sees an X where he targeted. The computer then shoots back , "NOOOOO!" screams Jeff as he realizes he has been hit. "BOOM!" Jeff shoots again, this time hitting a power up. Jeff is able to deploy an airstrike against his opponent. "DIIIIEEEEE!!!!" Jeff screams as he deploys his airstrike. "HAHAHAHAHA" Jeff laughs as he destroys one of his opponent's ships. He knows the ship is gone because a message is displayed saying "You sank the opponent's battleship." Jeff is glad he got his opponent's biggest ship. The computer misses Jeff's ships on it's next turn; however, Jeff is devastated to realize that a storm has started, and it's right where 2 of his ships are placed. Jeff hopes that his prayers to Poseidon everyday have not been in vain and his ships will survive, but Poseidon is not a merciful god. Poseidon punishes Jeff by destroying both his ships. Jeff now gets very mad and smashes the computer he was playing on.

Ana is an 80 year old who likes playing pickleball on weekends
Ana opens up the game and is prompted to select the difficulty. She chooses easy mode as she's trying to have a relaxing Sunday. She places her ships easily as the UI explains to her how to do it. She plays a fun game against the computer and wins easily. The computer lost many ships to storms and this made Ana very happy.

Sonar pulse

Jim is a 15-year-old student who doesn't like to lose. As a player, he wants to use the sonar pulse to reveal a portion of the map so that he can successfully sink the first enemy ship.
Jim opens up the battleship game and is prompted to select the difficulty. He selects a hard mode since wanted to try out something challenging and make himself busy in the meantime. He places his ships easily as the instruction explains to him how to play. The sonar pulse can show the status of the cell. Jim would witness a 3x3 diamond shape section of the enemy's board. This is a wonderful opportunity for Jim as he can see

three cells of the enemy's ship. The enemy lost many ships to the sonar pulse, this prompted Jim to give it another try since he won this hard challenge.

Captain’s Quarters

Hillary is a 21-year-old college student who never wins. She wants to use the captain's quarters to hit one cell of the ship that sinks the whole ship of the opponent's board.
Hillary decides to give the battleship game a try after her brother Jim recommended the game. He always wins in every game he plays so she knows that she is going to lose. She selects the difficulty to medium since she knows the basic instruction of the game. She places her ship diagonally and the computer gives her an error code instructing her to not place her ships diagonally or outside of the board. She notices that the battleship and destroyer's ship are armored and the instruction states that it takes two attacks on the same square in order to "hit". Unfortunately, the captain's quarters did not help her win the game, but she had fun.

Submarine

Max is an Accountant Manager who needed a break from work. He wanted to use the submarine so that he can take up to five blocks on the grid.
Max opens up the battleship game and is prompted to select the difficulty mode. He selects the medium mode since he doesn't have enough time and needs something more competitive. He places his ships easily as instructed. As a player, Max wants to use the submarine that can be placed on the surface or it can be submerged. It needs to be placed on the grid under any other type of surface ship and it occupies five blocks on the grid. He needs this feature so that he can successfully take up five blocks on the grid.

Space Laser & move command

Jim, the 15-year-old student who doesn't like to lose wanted to test the space laser and move command as a power up to win the game.
After winning the battleship game using the sonar pulse, Jim wanted to try out the space laser. The instruction stated that the space laser is fired from a network of geostationary satellites. As a player, Jim wanted to penetrate the water so that it would be able to hit both the enemy's surface ship and their submarine that was placed below it at the same time. He received the activation codes for the space laser only after sinking the opponent's ship. He was amazed at how he could hit the sub and the surface ship. The space laser was more effective than any other conventional bomb. He also noticed that

he can move his fleet when the command to move is given. He is able to move each of his ships to one position. He wanted to move his minesweeper but it wasn't moving. The game gave him an error code instructing him that he is unable to move a ship that is already at the edge. The instruction provided an alternative stating that he can undo and redo the previous move he made. He wanted to use the space laser so bad that he ended up playing the battleship game for four hours. His lack of sleep made him end the game but he was satisfied with his performance.

Airstrike

Jimmy jumps on the battleship and wants to see some fireworks, so he selects airstrike from his power-ups. He then selects a row he wants to attack and then laughs as he sees the entire row gets hit.

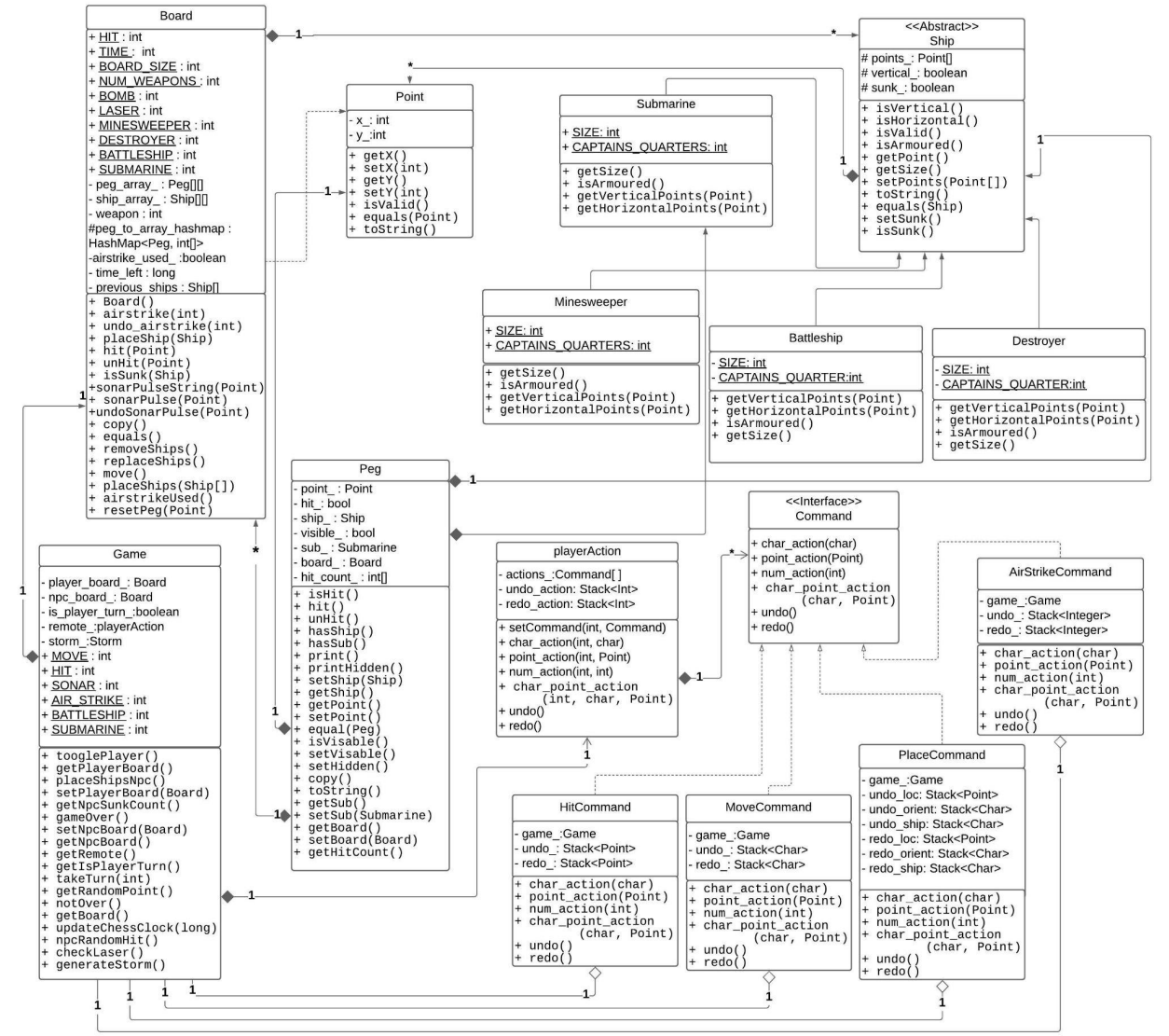
Storms

Jeff is playing and notices a storm has appeared randomly on his board. Fortunately the storm was not as big as any of the ships or they would have been sunk. Jeff gets pretty sad on his next turn though, because a massive storm occurred and sank his battleship because the storm was as big as the ship. Jeff then goes to pulse a location and is annoyed that the storm covered his pulse, so he cannot see where the enemy ships are in the pulse. Jeff notices small storms happen all the time, while bigger storms are rarer.

Gui hover effect

Bob goes to place his ships and sees a nice outline for the ship he is about to place as he puts his mouse over a square to place his ship.

Class Diagram



Design

The backend system contains 3 main parts: the game, the board, and the ships. We decided to not have a player class, as it did not make sense to us with the way we designed the board. We designed the board to have all the required information a player would need. This way a player is represented by the board they are playing on, which is very similar to how a game in real life would be played.

The board is a collection of our other main data type: peg. Pegs represent what pegs on a real life board would represent: what the players are hitting. So our pegs have information related to what ships they contain and how many times they have been hit. The board has all the pegs as a 10x10 array, so we make our board that way. When a board constructor is called, the board is created with all empty pegs. Pegs will have a ship once a ship is placed on them with the board's placeShip(Ship ship) method.

For the ships we decided to create an abstract class and then have each ship type extend the abstract class. This allows for polymorphism, which allows us to represent the ships as a ship array in the board class. The other added benefit is that we can pass a ship into any function and have it work still. Unfortunately, our ship design does have some flaws; we still have to check for ship type a lot of the time. This leads to our code becoming longer than it would have otherwise. To create a ship, the ship type is specified via the constructor, and then an orientation char 'v' or 'h', along with a start point are passed. The board will generate the "tail" of the ship and place it on the board. If it cannot be placed the method returns a negative value.

The game class has two boards, one for the player and one for the npc, and other information related to game state, but not board state. The game is responsible for generating storms and placing the ships on the npc board randomly. The game class is used extensively by our front end GUI.

The GUI relies on the game class to update what it will display to the user. In order to do just about any action on the board, the gui uses an implementation of the command pattern to talk to the game class. With this we were able to create a command that does something specific to the game class, call those commands whenever it was appropriate, and allowed us to keep track of the commands that we used.

The GUI is connected seamlessly to our backend logic by implementing the observer pattern. We create GUI observers which correspond to a button and location on the board. Whenever an action occurs on the board, the notifyAllObservers() is called and our GUI will update it's appearance. The update() function on each GUI observer is what determines the look of the board in the GUI.

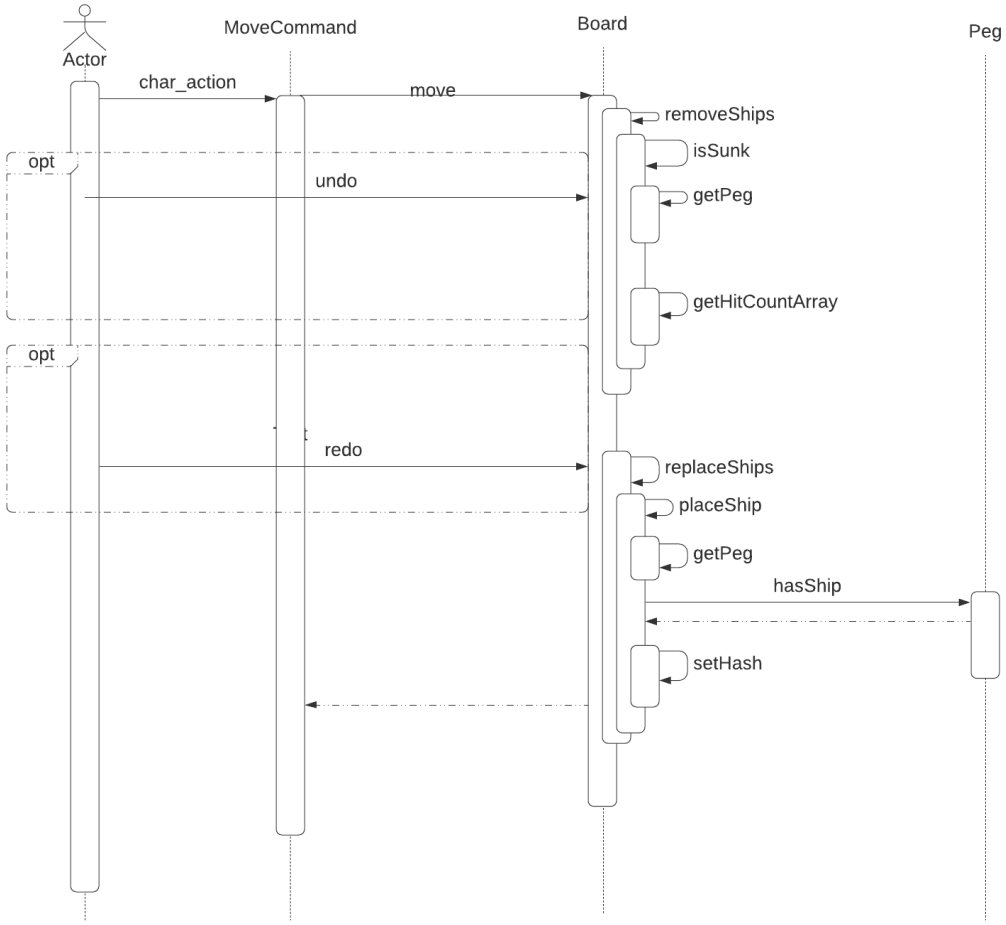
Example of how to place ships:

```
Board board = new Board();

Ship minesweeper = new Minesweeper( orientation: 'h', new Point( x: 0, y: 0));
Ship destroyer = new Destroyer( orientation: 'h', new Point( x: 0, y: 1));
Ship  battleship = new Battleship( orientation: 'h', new Point( x: 0, y: 2));
Ship submarine = new Submarine( orientation: 'h', new Point( x: 0, y: 3));

board.placeShip(minesweeper);
board.placeShip(destroyer);
board.placeShip(battleship);
board.placeShip(submarine);
```

Sequence Diagrams

User case

Jim, the 15-year-old student who doesn't like to lose. As a player, he wants to move his fleet so that he can use the undo and redo command.

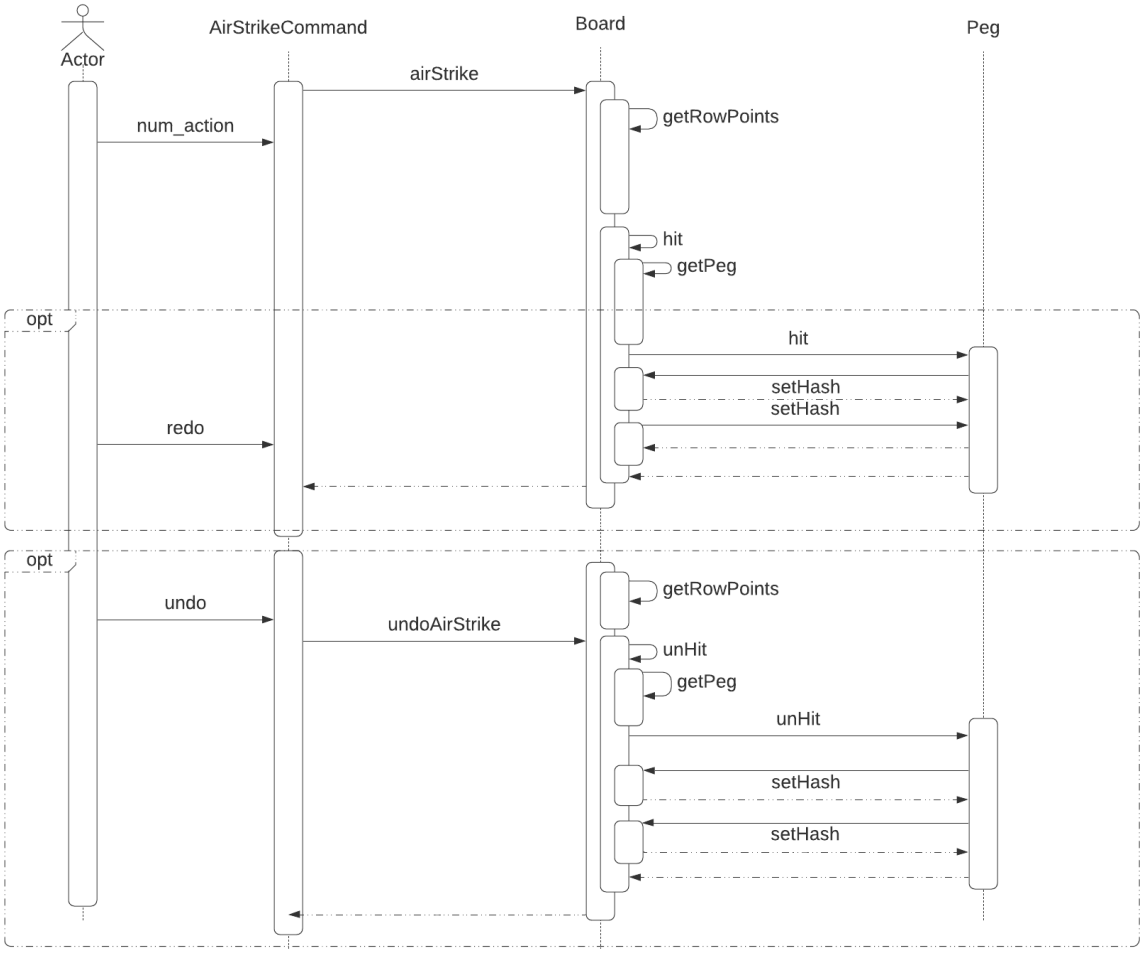


Move Command

The move command lets the user move their fleet. Each ship will move one position when the user specifies the direction they want to move. The char_action receives the direction the user wants to move, they can move North(N), South(S), East(E), and West(W). The move operation in the Board class can move the fleet and check if a ship is already at the edge. The user will also have the option to undo or redo their moves. In terms of user engagement, the GUI supports the move command by using the arrow key for direction.

User case

Jimmy jumps on the battleship and wants to see some fireworks, so he selects airstrikes from his power-ups.



AirStrike Command

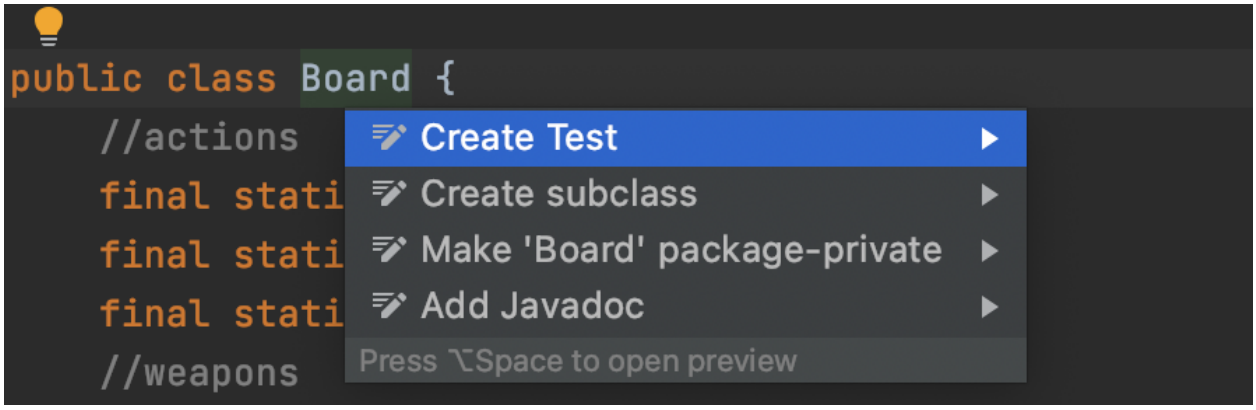
The airStrike command hits the entire row on the board with bombs. First, it gets the row numbers from the board using the getRowPoints. It then hits every peg by calling the hit() method on that row with a bomb. The user has the option to redo or undo their moves. The redo method calls the airStrike in the board class that is similar to the num_action. The undo method calls the undoAirStrike in the board class that gets the row numbers and unHits the row points. The airStrikeUsed boolean checks if the moves are redo or undo. The GUI supports the airStrike command by letting the user press the middle mouse.

Test

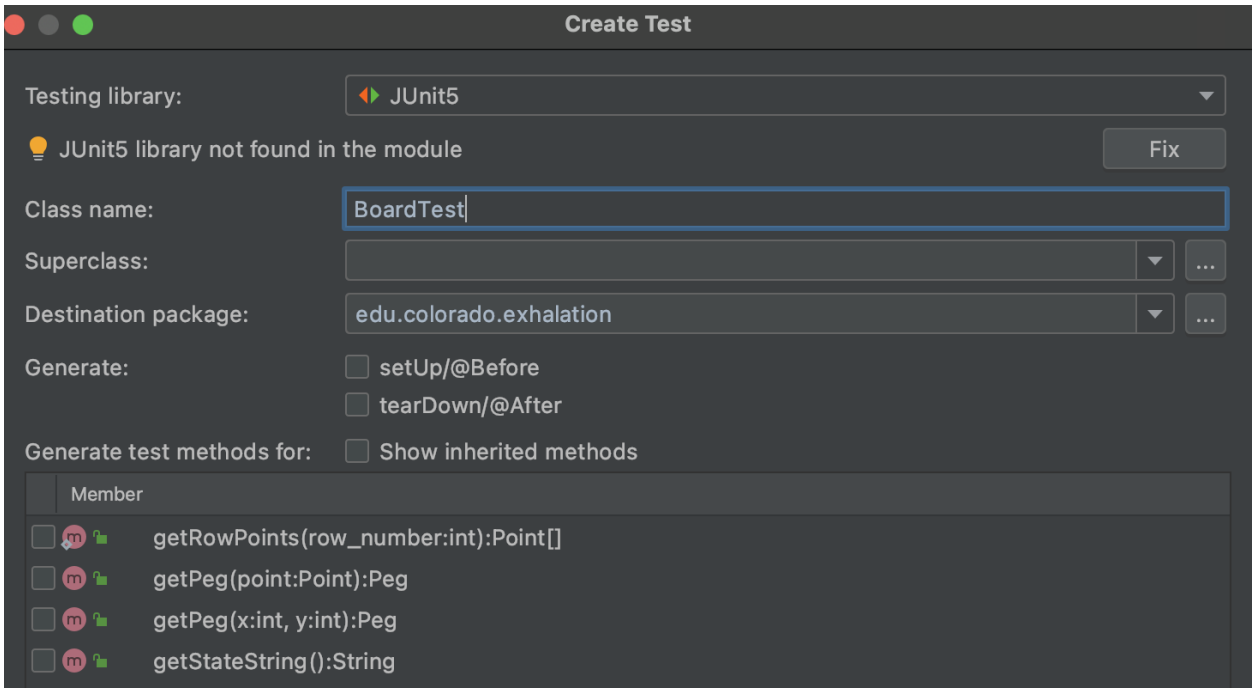
We used JUnit as our testing framework for this project. It is one of the best test methods for regression testing. It is used to write and perform repeatable automated tests as an open source application for JAVA. As developers, we were able to follow the test-driven methodology in which we were able to write and execute unit tests first before writing any code. When we are finished with coding our classes, all tests should be executed and passed. We re-execute all test cases everytime we add some code and make sure nothing is broken.

Create tests

1. Place the cursor at the Board class declaration and right click it. Select the Show Context Actions and from the menu, select Create Test.



2. Select the class methods that we are going to test.



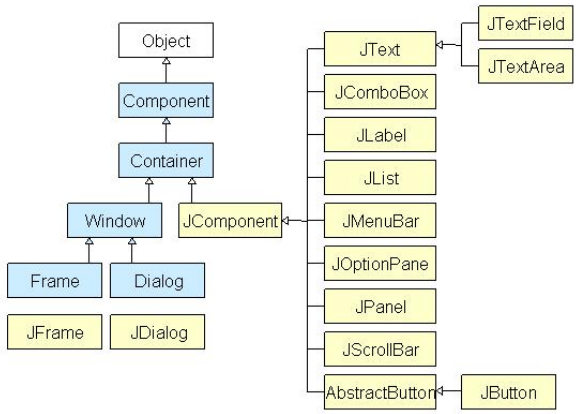
3. The editor takes you to the newly created test class, you can modify the testEmptyPrint() test as follows

```
@Test
void testEmptyPrint(){
    String empty_board = getEmptyBoardString();

    Board empty_test_board = new Board();

    assertTrue(empty_board.equals(empty_test_board.getStateString()));

}
```

The assertTrue throws an AssertionError (without message) if the condition given in the method isn't True

GUI

https://www.guru99.com/java-swing-gui.html



For this project, we used Java Swing as our GUI toolkit that includes the GUI components. Swing consists of a large number of widgets and packages that can be used to build sophisticated graphical user interfaces for Java applications.
We use “import javax.swing” in our program to use all the functionality provided by swing.
In the hierarchy diagram, it demonstrates that all components in Java Swing are JComponents. This can be added to the container class.

Some components we used for this project:
JFrame: A frame is an instance of JFrame, it is a window with a title, a border, a menu, buttons, text fields and other elements.

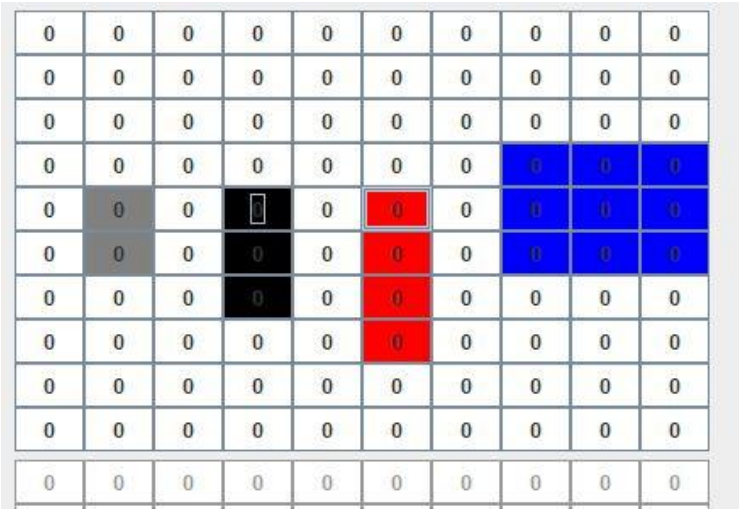
Example: `JFrame frame = new JFrame("Battleship");`

JPanel: A panel is an instance of a JPanel. For grouping elements, panels are useful and placed in the frame at suitable locations.

JButton: A button is an instance of JButton class. The class is an implementation of a push button, it has a label and creates an event when pressed.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;

//https://stackoverflow.com/questions/13787873/adding-buttons-using-gridlayout
public class Gui implements ActionListener, MouseListener, KeyListener {
    private JFrame frame;
    private JPanel player_panel;
    private JButton[][] player_buttons;
    private JButton[][] npc_buttons;
    static final int SIZE = 10;
    private int player_laser = 0;
    private int npc_laser = 0;
    private GridLayout boardLayout;
    private Game game;
    char orientation = 'v';
    private boolean pulsed = false;
```

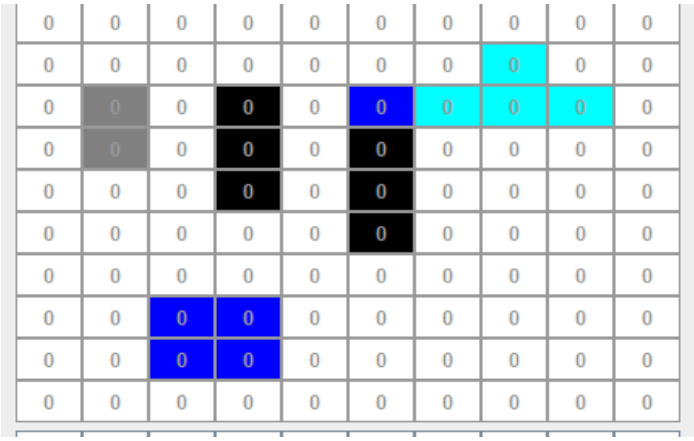
How to play



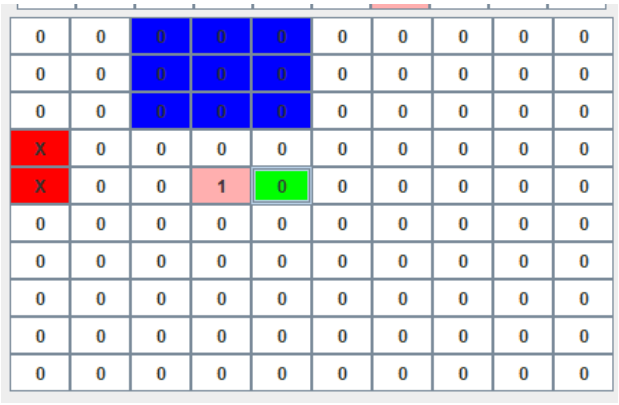
Grey ship - unarmoured

Black Ship - armoured

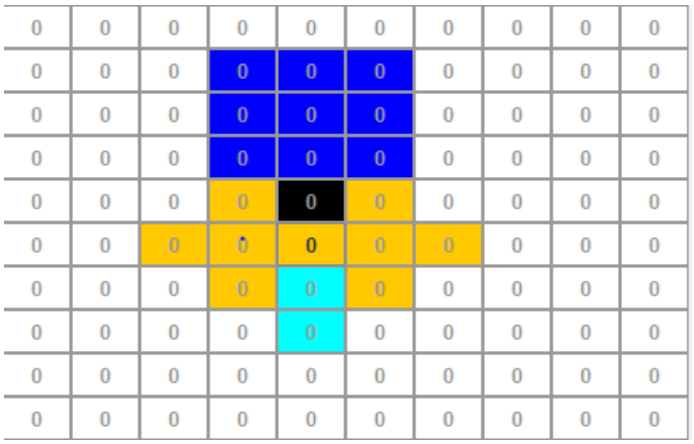
Blue - Storm or sub+ship (see below)



Light Blue - Sub

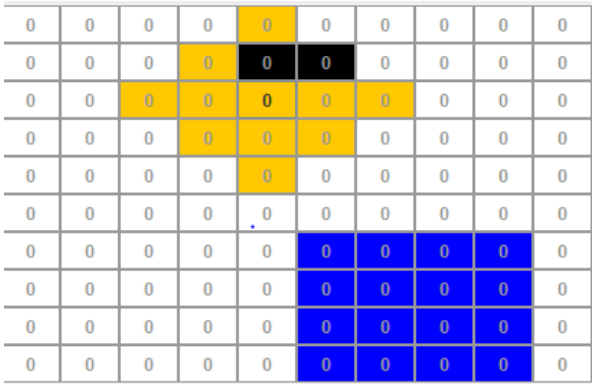


Red and X's- sunk ship
Pink- hit and missed
Numbers indicate number of bomb hits on peg
Green- current peg



Above is an example of sonar pulse usage. The yellow pegs are showing no ships on those pegs, while the light blue represents an enemy submarine. The black represents an enemy ship, but the blue is a storm obscuring the pulse's vision.

Yellow- empty pulsed pegs



Personal reflections

Jonathan Bluhm

I got a lot out of this project. I had never been exposed to TDD prior to this class, and at first the idea of it seemed very strange and I didn't like it. However, after being forced to practice it through this project I see the immense value in having unit tests. Refactoring is not really possible without the tests in the first place. So I learned a great deal about refactoring and how to write tests. Some tests I had to get creative with writing and that was a fun experience. I also learned a great deal about designing software systems. Due to the constantly changing requirements, we were forced to redesign certain elements multiple times from scratch. This got very annoying, but we eventually learned and created modular code. I very much now see the value in sitting down and thinking about design, because a bad design can cause various unnecessary problems.

Ethan Bilek

There were a good deal of things I learned/realized while working on this project. From the get-go I had no experience with Java at all, and being thrown into it headfirst helped me learn it quickly (and realize that with just a short adjustment period you can get used to just about any coding language that you haven't used before). On top of that, I had never learned/used Test Driven Development before and have come to see how useful it is. Often I'd put print statements in my code in different places to see where it was going wrong, but with TDD you can definitively see which classes/methods definitely work and it helps find where the issue in the code really is. The patterns were very interesting to learn about in class, and it was interesting to see which might apply to different aspects of the project and how it's not always necessary to use a pattern to solve a problem (and can even be a hindrance). I also realized something about myself and working with a team of random people: everyone's got their own schedules/desires on when they want things to be done. I found myself often not working on the code until closer to each milestone due date, which is something that I was fine with but others were not. That's something I need to change about myself in future projects so that it is easier for others to work with me and that it doesn't stress them out while working. Overall this was a very insightful project in respect to both programming and teamwork skills.

Ziad Alwazzan

I learned a lot while doing this project. To start, I had never worked with Java or JUnit. I have written unit tests before using c++ but never with a program/project this big. I learned the true purpose of test driven development and how much time it can save. Throughout this project, my group and I were constantly thinking about what design patterns we can make use of to modularize our codebase. At first, I was skeptical about changing our code, out of fear of breaking it. After multiple additions and changes we learned what we needed to change and became more comfortable with refactoring and rewriting code. I also noticed how planning before writing code would help. UML diagrams were a very useful tool that helped map out the classes and methods needed to perform a specific task. I consider myself lucky to have a group like this one. I felt challenged yet had a lot of help at the same time, and thanks to partner-programming, teamwork was so much easier.

Yosan Russom

When I started working on this project, I was not familiar with Java and I have never used any testing framework. Collaborating with my team gave us a good opportunity to exchange ideas and come up with creative features for this battleship game. This project gave me an insight on how to produce professional-quality design documents. I learned how to demonstrate an understanding of a user's expectation and standards that are suitable for communication in this field. I have experienced different types of model systems such as UML or entity-relationships diagrams for this project. I really enjoyed working with my teammates. Participating in the development of the battleship project in a team environment has taught me on how to become an effective team player and the importance of design patterns.