Battleship

Project Documentation

Team: Caterpillars

Members: Bradley Ma, Maksim Vasilyev, Jaskrit Singh

BATTLESHIP
AGE 7+
ADULT ASSEMBLY REQUIRED • DOIT ÊTRE ASSEMBLÉ PAR UN ADULTE • REQUIERE SER ENSAMBLADO POR UN ADULTO • DEVE SER MONTADO POR UM ADULTO
Hasbro Gaming

Image Source: https://www.amazon.com/Hasbro-A3264-Battleship-Game/dp/B00C0ULS3G

Project Description

Battleship is a classical turn based board game that utilizes a player’s deduction skills in order to sink the opponent’s ships before they sink his or hers. The goal of this project was to first structure and implement this game in Java with an object oriented approach. With this base structure, our team decided to bring the game to life inside an android app, which contains simple yet clear graphics and an understandable user interface. Of course, in order to improve our game from simply being the same from the original board game, we have also implemented several new features we hope that players will enjoy and make the game more difficult yet also more interesting.

Development Process

Our team's approach to successfully unite and work efficiently was to utilize a modified version of the Extreme Programming (XP) process.

The first core aspect of XP was the using the process of test driven development (TDD). Under TDD, our development process in terms of actual coding began following this cycle: (1) write unit tests that clearly expects one thing from the implementation, (2) if the test fails (it should) implement the source code to make it pass, (3) commit these changes and push to the shared github if appropriate, (4) go back to step (1). TDD is an effective and straightforward method to guide the direction of our implementation. It also enabled our team to gradually build up a strong foundation of tests, which protected us from inadvertent mistakes, such as from faulty refactoring or newly implemented features. Due to the nature of the course requirements for each project milestone (i.e. new features, refactoring examples), it was imperative for us to maintain a solid basis of unit tests to ensure that new functionality would not interfere with previous work.

The second core aspect XP was the usage of pair programming. During our weekly meetings, we would first discuss the goals and expected requirements for the next milestone, and then would split off into pairs. In each pair, one member will do the actual coding, while the other pair watches for mistakes and provides guidance. This is not only more effective, as we now have two perspectives of a certain block of code, but it allows for each member to have shared exposure to a certain part of the project, thus collective code ownership. This allows for faster communication as it prevents the need for other members to waste time in explaining their own code.

With these aspects of XP mentioned above, refactoring our code became easy. In order to modify the inside logic of our code without altering the outcome, it is imperative to have unit tests to ensure successful refactoring, as well as

An additional aspect of XP we used was the agreement of a coding standard. This standard was set before we even began developing code. This enables our code to be consistent in style throughout our project, which keeps it organized and readable.

With the utilization of Slack, our team was capable of communicating constantly on updates and issues as a team outside of our meeting. It kept everyone up to date and helped organize upcoming meetings in face of scheduling issues.

After migrating to an Android project in the middle of the semester, we had to utilize a new framework to allow for android testing. Robolectric allows tests to be run in a simulated Android environment inside the JVM, without the need for an actual device or emulator.

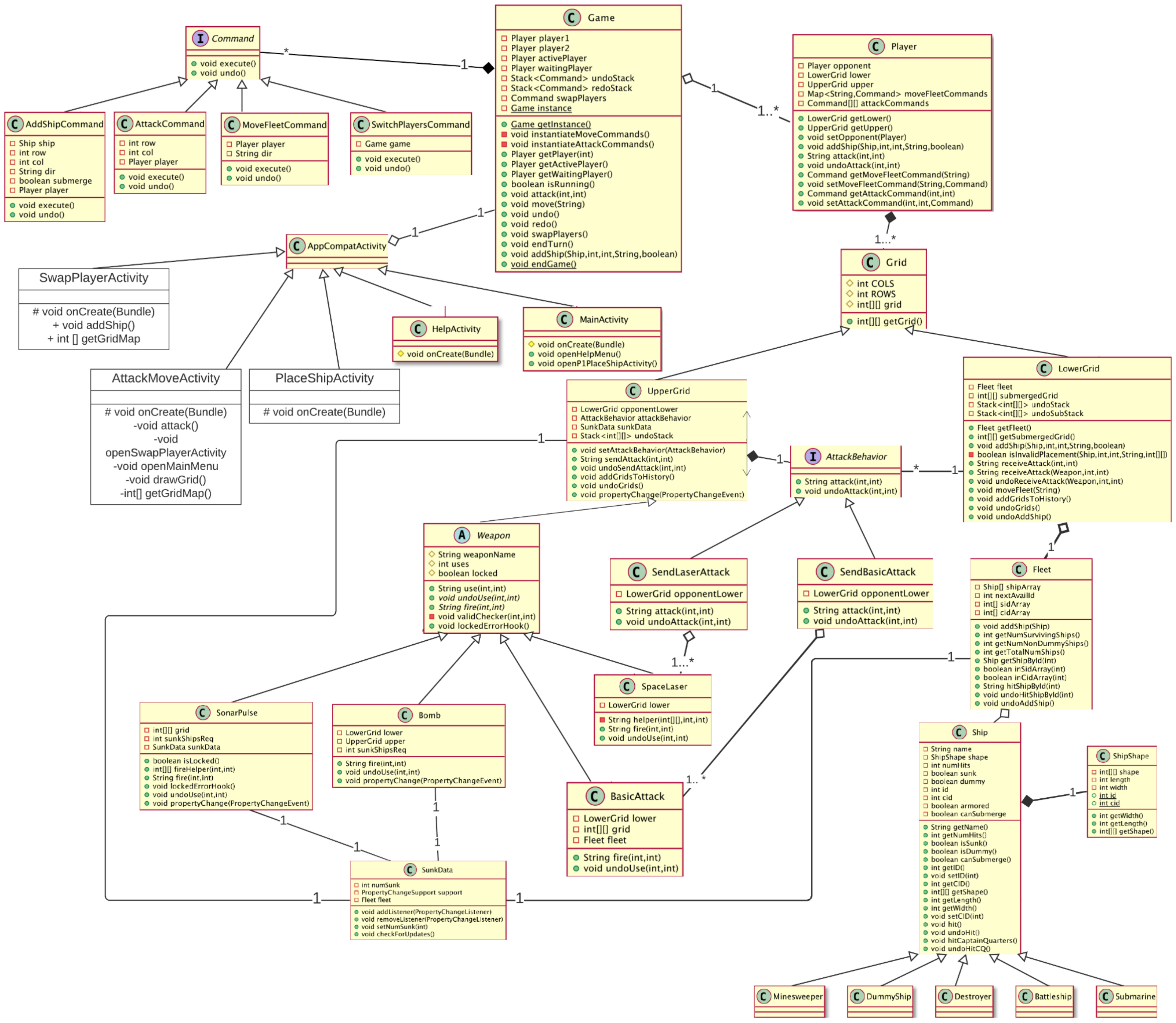
Requirements and Specifications

User Stories

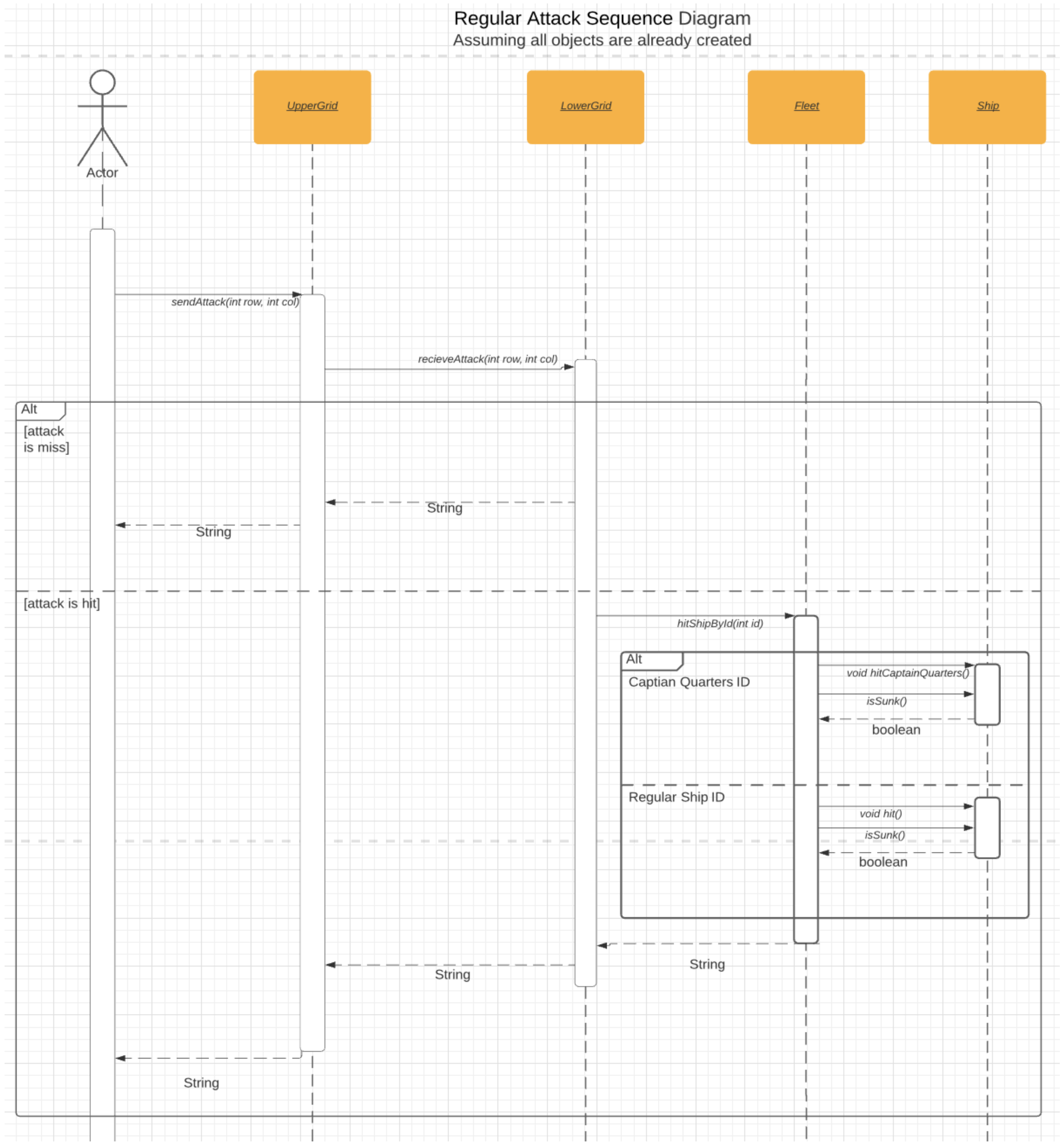
- ❏ As a player, I would like to choose where to put my ships, so that I have control of the game.
- ❏ As a player, I would like to choose where I want to fire an attack, and learn the outcome of this attack, so that I can deduce where to attack next in order to win.
- ❏ As a player, if I attack a ship at its captain's quarters, I will receive a "MISS" message if it is armored, or a "SUNK <shipname>" if it is not.
- ❏ As a player, when I reach all requirements to use the sonar pulse, I am able to use this new weapon in order to receive info about my enemies fleet location.
- ❏ As a player, I can place a submarine that is on the surface of the water or submerged, which allows me to place a submarine below another ship.
- ❏ As a player, when I reach all requirements to receive the space laser code, my attacks will now be using the space laser, which will hit both surfaced and submerged ships at a coordinate with the same shot.
- ❏ As a player, I would like to move my entire fleet in a certain cardinal direction, which increases the difficulty for my opponent to sink all my ships.
- ❏ As a player, I can utilize a new weapon called Bomb, which allows me to attack in a "+" shape across the board of 3 tall and 3 wide.
- ❏ As a player, I would like to be able to place down a new ship called the Dummy, which tricks the opponent to waste time looking for an actual ship.
- ❏ As a player, I would like to be introduced to the menu of the game through an android phone app, which allows for easy visualization and simple gui's.
- ❏ As a player, I would like to be introduced to the ship placement screen of the game through an android phone app, which allows for easy visualization and simple gui's for me to place my next ship on my grid.
- ❏ As a player, I would like to attack my opponent through the app and be presented with straightforward screens that allow me to do this, which allows for a more friendly and visually pleasing experience.

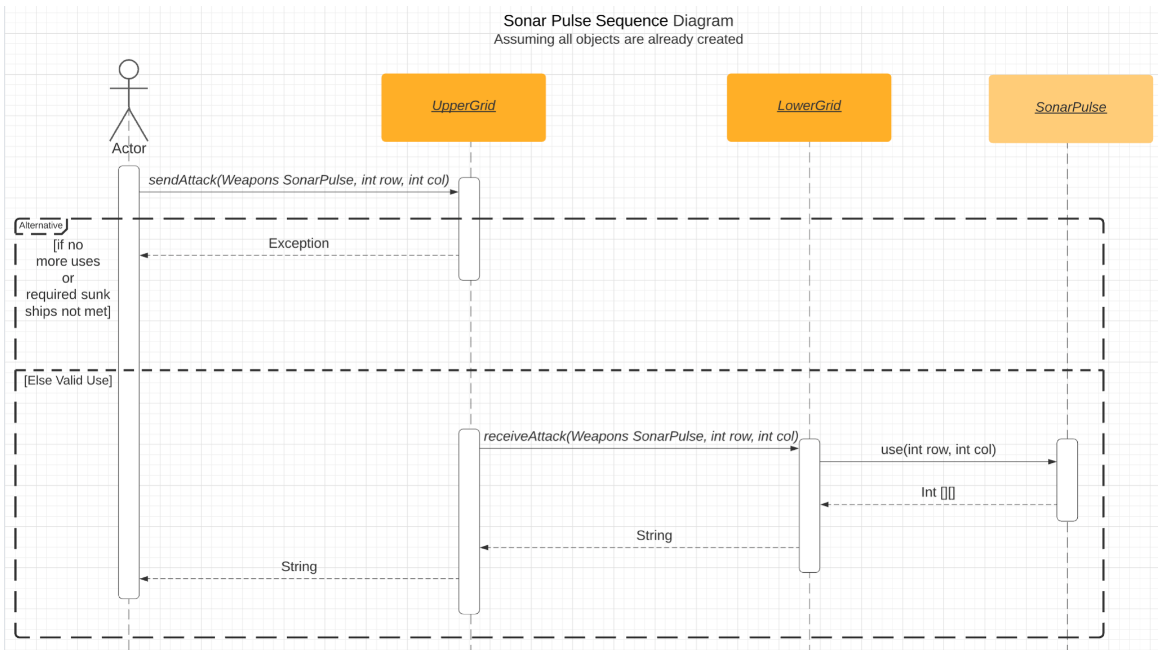
Architecture and Design

Class Diagram

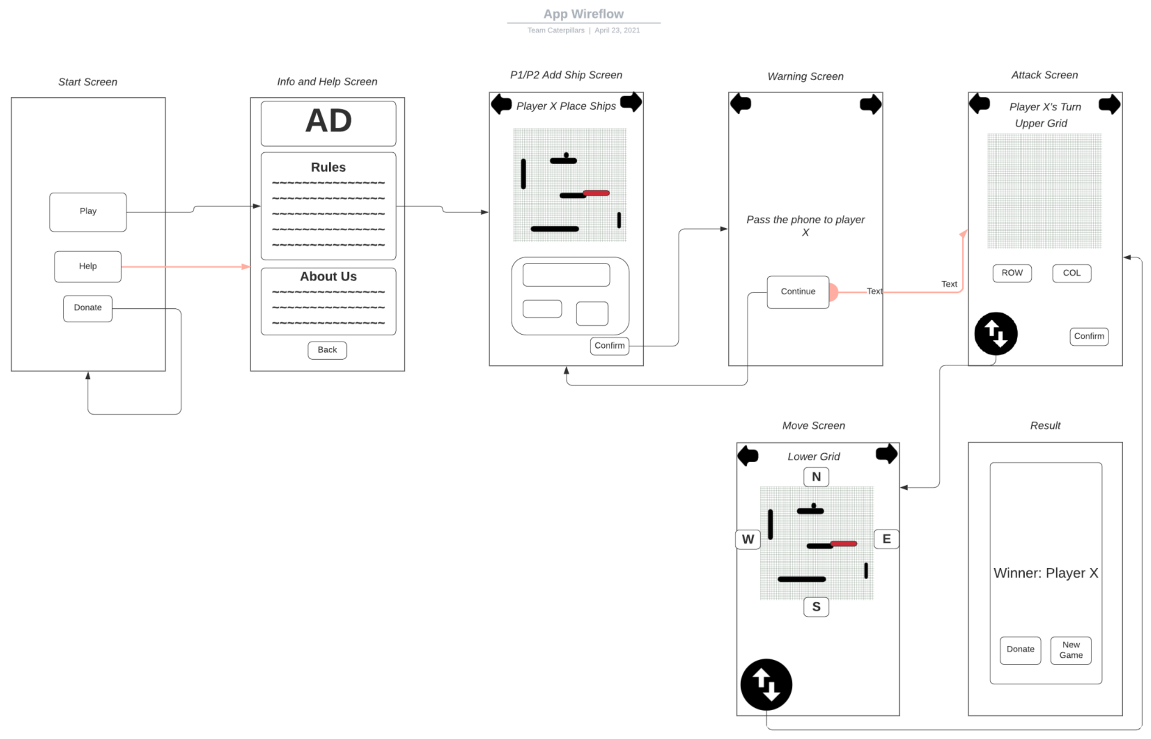


Sequence Diagrams




Sonar Pulse Sequence Diagram
Assuming all objects are already created
Actor
UpperGrid
LowerGrid
SonarPulse
sendAttack(Weapons SonarPulse, int row, int col)
Alternative
[if no more uses or required sunk ships not met]
Exception
[Else Valid Use]
receiveAttack(Weapons SonarPulse, int row, int col)
use(int row, int col)
Int [][]
String
String

WireFlow


App Wireflow
Team Caterpillars | April 23, 2021
Start Screen
Play
Help
Donate
Info and Help Screen
AD
Rules
About Us
Back
P1/P2 Add Ship Screen
Player X Place Ships
Confirm
Warning Screen
Pass the phone to player X
Continue
Text
Text
Attack Screen
Player X's Turn
Upper Grid
ROW
COL
Confirm
Move Screen
Lower Grid
N
W
E
S
Result
Winner: Player X
Donate
New Game

Big Picture

(Note: The class diagram above is a great visual representation of the top-down structure of our system)

Our system is ultimately encapsulated inside an instance of a Game object. This object is maintained through a **Singleton pattern**, which ensures the same Game is run throughout our app. This object then owns two instances of a Player object. The Game object alternates the turns between each player, maintains a history of commands, and keeps track of the game state. The commands are implemented using the **Command pattern**, which enables for simple undos. The two Player objects then interact with each other, which enables the gameplay of each turn in Battleship. Each Player has two instances of the Grid object, one UpperGrid and one LowerGrid. The UpperGrid tracks the position and results of its respective Player's attacks. The LowerGrid maintains and tracks the position of the respective Player's fleet of ships, as well as the position and results of the opponent Player's attacks. Each Fleet owns a collection of Ships, and facilitates the communication between the LowerGrid and the Ships. Each Ship holds its own information, such as its name and its shape.

Player's can send attacks to the opponent through their own public methods that abstracts away from the respective methods in the Grid classes. The Player's BasicAttack can be upgraded into the SpaceLaser utilizing the **Strategy pattern**, where the AttackBehavior is modified after a ship is sunk. Player's can also utilize other weapons (SonarPulse, Bomb) that must be unlocked after a number of ships have been sunk. These weapons and the SpaceLaser utilize the **Observer pattern**, which are subscribed to the SunkData Observable. The SunkData communicates with the respective Fleet, and indicates to all of the subscribers when a ship is sunk.

All four weapons utilize the **Template method pattern**, as the template method is defined in the base Weapons class, and defines the basic order of execution. It allows for the different weapons to modify and override any specific implementation that is unique to that weapon.

Ultimately, our Android app utilizes our various Activity classes, which instantiates or gets the current instantiation of our Game, and utilizes its public methods as a response to the actual players' interaction with the app's GUI.

Personal Reflections

Bradley Ma:

With all things considered, I enjoyed working on this project. I think it was a great tool to apply the design patterns we've discussed in class as well as get more experience with a new language and frameworks I wasn't familiar with. However, I feel like this project could come with many improvements. The review meetings could have been a bit longer, even though fifteen minutes looks fine on paper, an extra five minutes could have given us better feedback from the instructors. Also, while I do understand the purpose of vague requirements to mirror those in industry, I think it counteracts with the true purpose of the class, to learn design patterns. I think our team wasted a lot of time trying to decipher and interpret the specifics of each new feature, and if they were even a bit more clearer, less vague, I think it would have been a much better use of our time and semester. Something that puzzles us is very specific Java issues that we ran when trying to migrate to an android app. Unfortunately, we did not have enough time to dive too deep into discovering the truth with these issues in class. Overall however, I enjoyed this project. Some main lessons I learned by working on this project is the importance of communication and being on the same page as my team members, as well as the power of test driven development.

Maksim Vasilyev:

I will start my reflection by saying that I thoroughly enjoyed working on this project. I believe that our group was well balanced with talent and personality, and that we were able to develop a solid product. Many of my gripes with this project are due to the online format of the course. I would imagine that if the course was in person, we would have been able to conduct the review meetings in person, which would clear up nearly all of our project confusion as there would be more time in the review meetings. I believe that the openness and vagueness of the requirements was a good thing, but the problem was that we were not able to answer our initial questions for at least a day. If the class was in person I believe this problem would have been avoided. I wish that we had more time to work on the second leg of the project, as well as a little more freedom to focus on our android development rather than having to implement a new game feature every milestone. Overall I believe this project was a great platform to teach me about design patterns and I really enjoyed working with my group.

Jaskrit Singh:

I think this project was a good playground with which to test out different design patterns and other object oriented design practices. I learned a lot about how to do test driven development and how helpful it can be when numerous changes have to be made to the code. I also got more confident using github after having worked on this project, with many different issues popping up like merge conflicts, restructuring of directories, and files getting pushed to GitHub when they should not have been. I think the initial few milestones were good practice for figuring out how to make code work for changing requirements. That being said, I do wish these requirements were a little bit more clear, as there were many different ways to interpret them and we spent a lot of time discussing which way we should do it. I think that the milestone meetings were very helpful for getting advice on our project from someone with more experience, but it would have been nice to have just five more minutes for each meeting so that we have a little bit more time to discuss in more detail. I wish we could have spent more time on the app portion of the project and tried out some more design patterns specific to the app instead of having to balance it with the addition of other requirements.

Installation

Our project utilizes the Android SDK, so this must be installed

Our project can be run either through an android emulator, or through an android device on developer mode. One of these must be available before running our project.

Our app can be (potentially, still setting up) downloaded through the app store using the link available in the README file at our github repo.
Note, your email must be whitelisted in order to be allowed to download.

Issues

1. Robolectric (Android test framework) has a current opened issue when running multiple tests files with coverage. This can be avoided when running all tests with coverage in the src code folder, and **not** the test folder.
2. Many issues our team had when attempting to migrate to an Android environment through IntelliJ was pushing configuration files to the github that would affect the Java version of our own individual workspace. We spent a lot of time tinkering with IntelliJ's project settings just to avoid build errors.