Battleship

Project Documentation

Team *Pomegranate*

Nikhil Kishore, Jeffrey Wentz, Taylor Fahlman, Dylan Camus

Preface

This documentation contains the overview and all technical details of the implementation of the Battleship game in Java. This documentation is designed to be a reference material, which you would follow along to get an idea of the project and the code implementation. Therefore, this document is designed to augment the code and comments, not replacing them. Please refer to both of them when trying to understand the project.

Table of Contents

Development Process

In the course CS 361, our group, team pomegranate used extreme programing (XP) in our development of the game Battleship. Extreme programming is a way to efficiently develop software. It is low risk, flexible, predictable and scientific. It makes developing software much easier. This is done by planning an approach, creating a flexible schedule and working in teams with strong oral communication. The four basic activities our team followed during the devolvement Battleship was: iterative development, refactoring, testing, and collaborate development.

Iterative Development

Creating a large program presents it's own challenges.The incremental development process broke down our primary objective into small manageable parts. It allowed us to avoid project risks by simplifying the entire project and provided more ease-of-change during the development process. Looking back through the design process we followed, incremental development really helped make the project run smoothly. Without following this process, we could have easily spent extra man hours working through errors in our logic and having functionalities that did not cooperate with others. Following a sound design process is something I would suggest to any team setting out on creating software.

Refactoring

Programmers restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility. Refactoring was a big deal when we started off with having procedural code. In order to make it more object oriented, we had to refactor. Refactoring also helped us make our code cleaner and less messy. This made our code look much better than before, since functions were simplifies and classes were downsized. One problem we had while refactoring was failing test cases. Changing up the code caused test cases to fail, so we to change them back. Not all code is refactorable, sometimes you have to leave the implementation how it already is.

Testing

The testing phase was right after we had completed designing our diagrams. This was done before the actual implementation of the game classes/object, since the test tells you when you are done coding. If the test runs properly and passes, that means that you are done for the moment, and when you can't think of any tests to write that might break, you are completely done. Testing was a great way of seeing the implementation of our classes was working or not. Tests are both a resource and a responsibility. You don't get to write just one test, make it run, and declare yourself finished. You are responsible for writing every test that you can imagine won't run immediately. And this is exactly what we achieved during our testing phase. Through testing we found that our tests did not always match up with those of our instructor’s. This just showed us the importance of clarifying aspects of the requirements and double checking to make sure you and the customer have the same picture in mind when creating software.

Collaborative Development

All production code is written with two programmers at one machine. With a group of four, we split into pairs and worked on different part of the project. One person was coding while the other gave ideas and monitored. This practice went well because everyone has his or her own strengths. Some of us were not good or familiar with Java and some of us were not. We ran into a few issues when trying to work in a collaborative development setting, the main of which bring meeting out time estimates. Working in a group slowed down our development time which lead to scheduling more hours during the week in which to meet. While it was slower, it also proved to be helpful when tackling issues early on in the development phase as it was likely one of the members would catch it.

Requirements and Specifications

Place

- Use Case Brief: This happens in the beginning of the game. For this stage, the player selects one of his ships, then selects a place on the board to place that ship. It can be placed either vertically or horizontally. Once placed, the player cannot place any other pieces there.
- User Story: As a player I want to place ships on the board in valid locations.
- Casual Use Case: Place Ships At the start of the game, each player chooses the locations of his fleet. The player may place his ships anywhere on the grid as long as the ships orientation is either horizontal or vertical. In addition, ships may not overlap and no part of any ship may be outside the grid. Each player begins with four ships in their fleet.
- Fully Dressed Use Case:
    - Place ship
    - Primary actor: Player
    - Goal in context: Choose locations on the grid for a players fleet.
    - Scope: Game setup
    - Level: User-goal
    - Stakeholders:
        - Player: Place ships in unique locations in order to avoid fleet destruction.
        - Opponent: Place ships in unique locations in order to avoid fleet destruction.
    - Preconditions: None
    - Guarantees: All ships for both players are placed in valid location on the grid.
    - Triggers: Game startup
    - Success scenario:
        - 1. Game is launched
        - 2. Player is prompted to choose a location for one of his ships

        - 3. If the target location contains no other ships, is on the grid, and is in a horizontal or vertical position, the ship is placed.
        - 4. Ship placement continues until all ships are placed
        - 5. Game setup ends Extensions: If the player chooses an invalid location, he is asked to re-input a correct location.

Attack

- Use Case: This is the first stage of a player's turn. For this stage, the player selects a place on the opponent's board to hit according to a coordinate. Once the place is selected, it is determined whether or not there has been a hit.
- User Story: As a player I want to target coordinates on the opponents playing field to attack.

Captains Quarters

- Use Case: When an attack is called, it is determined if the spot hit is a captain's quarters. If it is, then it is determined if the space has been hit before. If not, it is returned as a miss. If so, the ship is sunk. Then it is determined if the game is over.
- User Story: As a ship I want to have Captains Quarters, which will be my major weakness.

Sonar Pulse

- Use Case: When sonar is called, first it is determined if the user has at least one more sonar charge left. If not, the sonar doesn't happen. If the player has sonar charges left, then the player selects a spot to deploy the sonar. The location is validated, and if valid, the player can see the area around the location of sonar within the bounds of the board.
- User Story: As a player I want to be able to send out a Sonar Pulse, which reveals multiple enemy tiles.

Submarine

- Use Case: The submarine has the ability to be placed on the surface or submerged. When submerged, it has the ability to be place under any other ship. It takes up five blocks on the grid.
- User Story: As a player ship I want to place the submarine submerged because that way normal ships can't attack me.

Space Laser

- Use Case: The space laser is available to the player after sinking the first enemy ship. The laser is fired from a network of geostationary satellites, which allows it to hit both surface ships and the submerged submarine.
- User Story: As a player I want to be able to use the space laser to attack most the submerged submarine and the ships on the surface.

Move Fleet

- Use Case: When the player calls the move action, every ship in the fleet moves one position. The player will specify the direction of move (N, S, E, W). If a ship is on the edge on the grid, it will not move if the direction is towards the grid.
- User Story: As a player I want to be able to move my fleet whenever possible to avoid enemy hits.

Undo/Redo

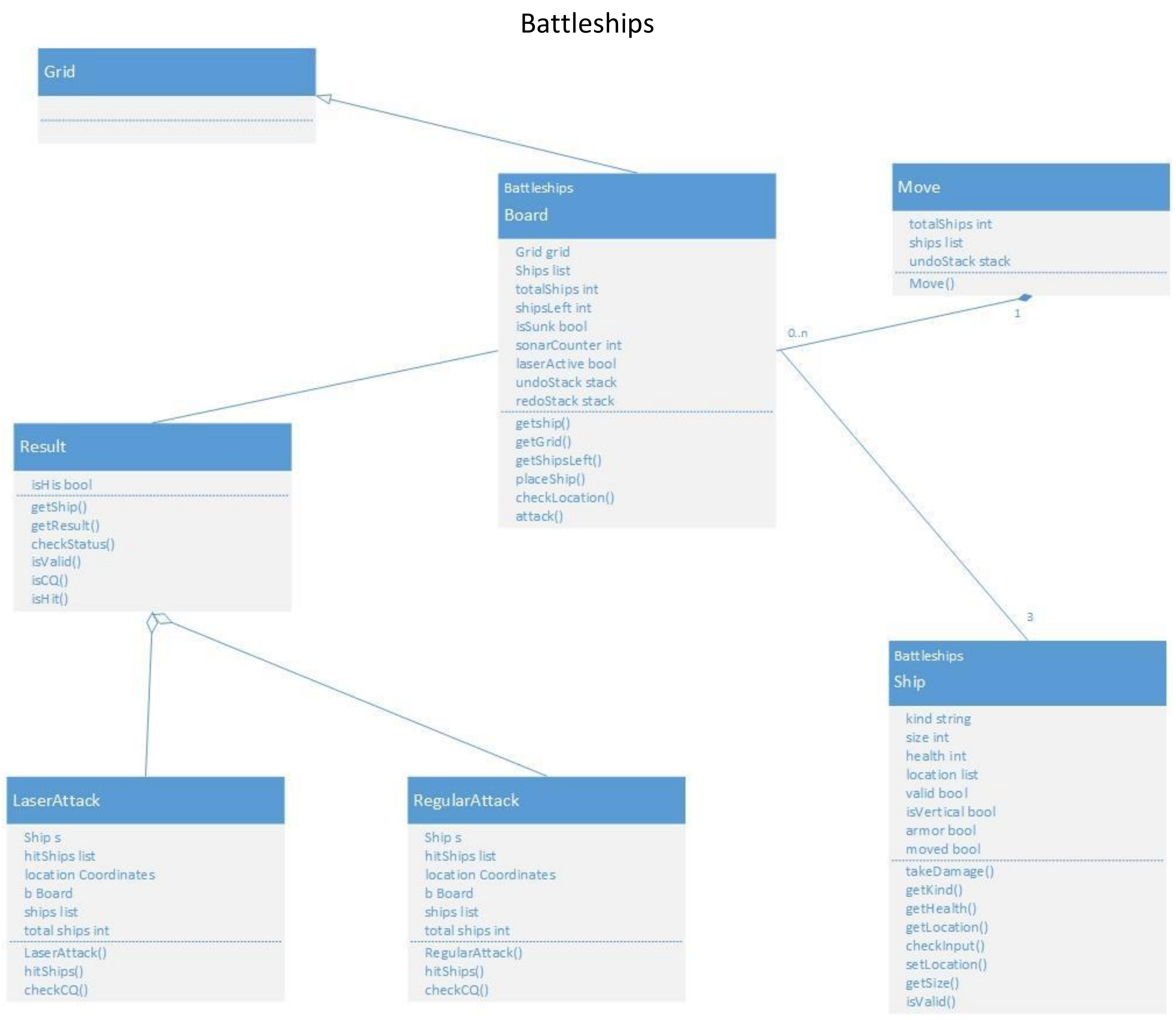
- Use Case: After the player has made a move, the player has the option to undo or redo the same move. If the player chooses undo, the action will be reverted. IF the player chooses redo, the action will be carried out again.
- User Story: As a player I want the ability to undo or redo a move that I've just carried out.

Architecture and Design

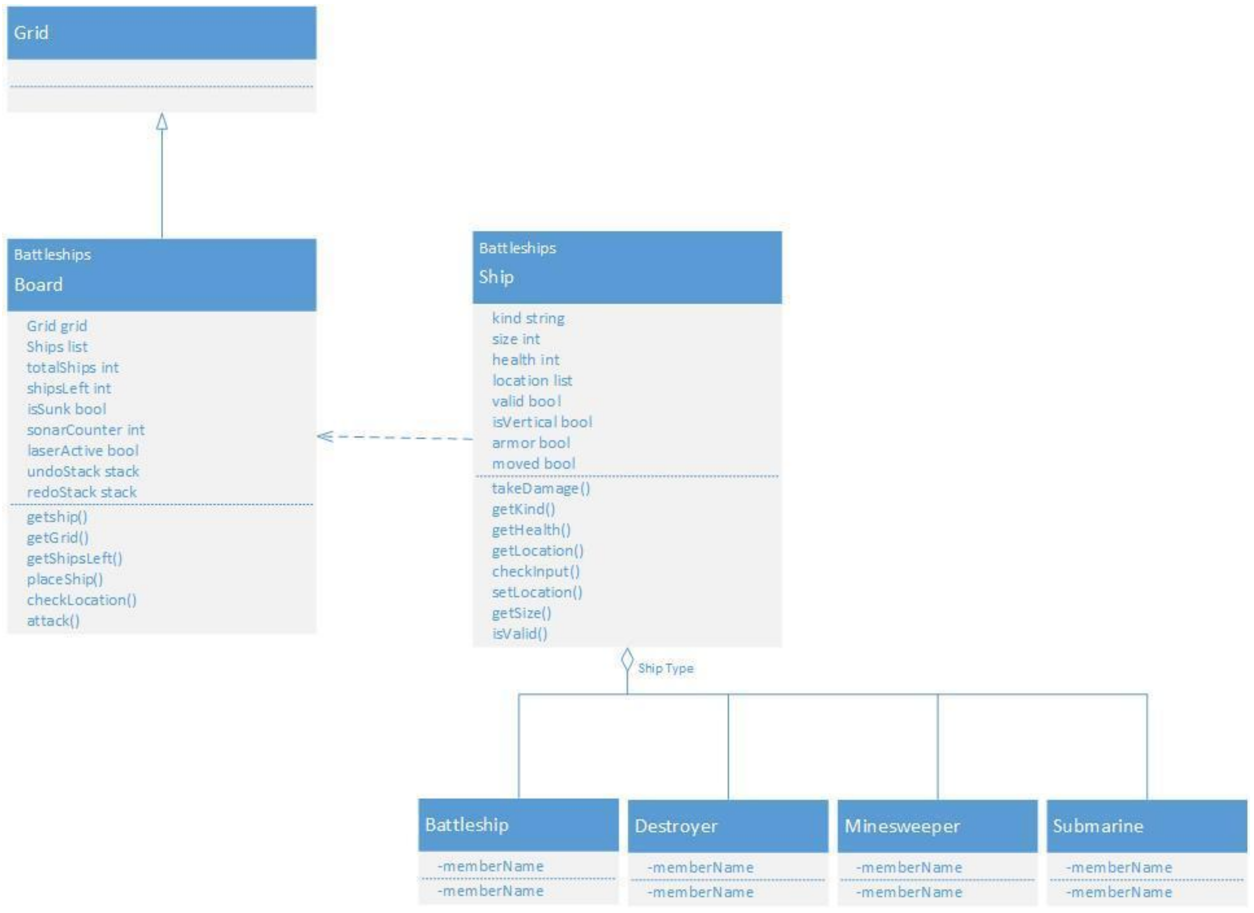
System Description

The system is broken up into small sections that fit together. At the center of the system is the Board class. The board is where the ships are set and where the ships can be altered with actions like move and attack. The ships are all different objects which derive from a superclass ship, and each respective ship will set its information and also store its location on the board. Particular parts of the board, such as move, attack and sonar and either functions or classes related to board. The last few classes are things such as Status, Result, and the like are all individual classes that are used in other classes but exist separately.

Class Diagrams



Ships


Grid
Battleships
Board
Grid grid
Ships list
totalShips int
shipsLeft int
isSunk bool
sonarCounter int
laserActive bool
undoStack stack
redoStack stack
getship()
getGrid()
getShipsLeft()
placeShip()
checkLocation()
attack()
Battleships
Ship
kind string
size int
health int
location list
valid bool
isVertical bool
armor bool
moved bool
takeDamage()
getKind()
getHealth()
getLocation()
checkInput()
setLocation()
getSize()
isValid()
Ship Type
Battleship
-memberName
-memberName
Destroyer
-memberName
-memberName
Minesweeper
-memberName
-memberName
Submarine
-memberName
-memberName

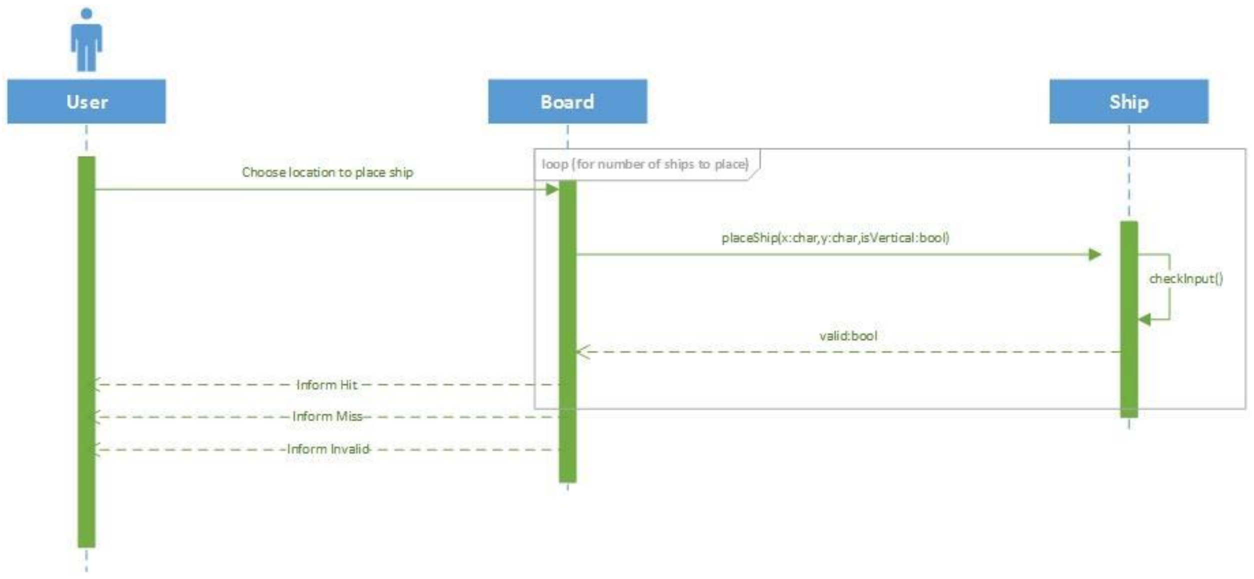
Sequence Diagrams

Use Case:

Actor: User

Goal: Place a ship on the board

Brief: The player selects one of his ships, then selects a place on the board to place that ship. It can be placed either vertically or horizontally. Once placed, the player cannot place any other pieces there.

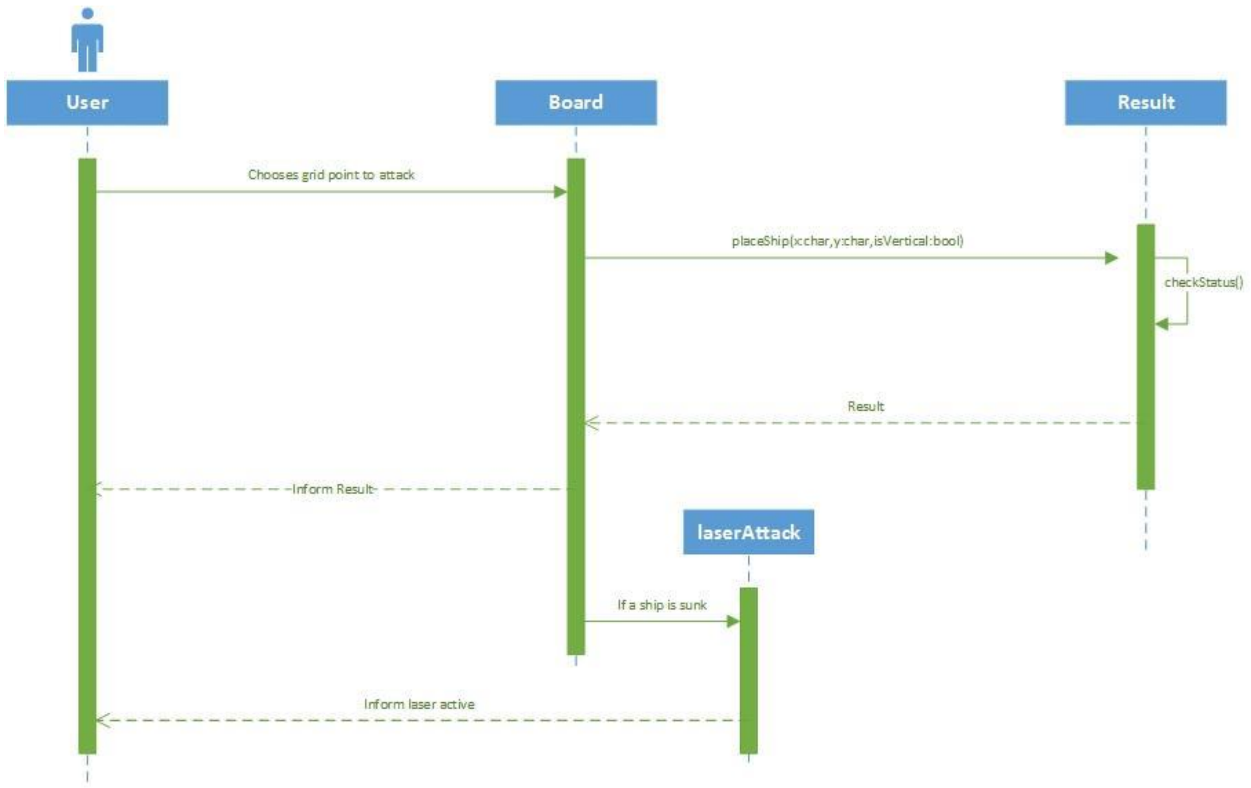

User
Board
Ship
loop (for number of ships to place)
Choose location to place ship
placeShip(x:char,y:char,isVertical:bool)
checkInput()
valid:bool
Inform Hit
Inform Miss
Inform Invalid

Use Case:

Actor: User

Goal: Attack a location on the board’s grid

Brief: The player selects a place on the opponent's board to hit according to a coordinate. Once the place is selected, it is determined whether or not there has been a hit.



Personal Reflections

Taylor Fahlman:
Overall this was a very good learning experience for me. While I had basic understanding of Java and OOP before this class, this project really helped me grasp OOP in a much better sense. For example, I learned how to implement the Command Pattern, and it helped OOP to make more sense to me. Also, I had heard of TDD, but this assignment helped me to have a solid grasp on it and see the benefits of such a development process. This battleship program has really expanded and improved my programming ability as well as my communication skills.

Jeffrey Wentz:
Going into this class I had very little experience when it came to Eclipse and Java. This project introduced me to the Java world and taught me a lot of the features Eclipse has to offer. This was also my first time using a test driven development process and am glad I was put in a situation that required it. I’ve worked on projects with teams before, but this was the first time we’ve had to meet deadlines and present our work on a bi-weekly basis to our superiors. Creating class and sequence diagrams proved difficult, but really help others when they’re trying to understand a program's architecture that they didn’t help produce. Looking back on this class as a whole I’ve definitely grown from the experience and will be able to relate what I’ve learned to real-life situations.

Nikhil Kishore:
Going into CS 361, I had very little experience with creating use case diagrams, using Eclipse and Java. I had taken CS 362 before CS 361, so I knew a lot of the back end work, but not much of the front end. I thought this class was a great experience for me, since I learned how to actually begin a big project. Such as making a plan and how to work in groups. I thought doing TDD and pair programming helped me improve working in groups and managing tasks. The

class also helped me improve on object oriented programming, since I am very used to procedural coding.

Dylan Camus:
I found this class to be the most helpful class I have taken so far at OSU. I learned so much about how software projects are actually conducted in the real world. I had no previous experience with test driven development or Extreme Programming, but now I feel very confident with these skills. I learned quite a bit about object oriented programming and how to use design patterns to clean up code and make it much more readable. I also learned a lot about pair programming and working on a deadline, as well as working on a project as a team in general. In addition, I learned Java as well as Eclipse and JUnit. I also learned how to use git and github. I feel that this class has prepared me well for what I should experience in a career in computer science.