Battleship Documentation

Isaac Vance
Stefan Primelles
Jarek Carlson
Neel Karsanbhai

Table of Contents

Description

This document contains an overview and relevant technical details of Battleship. This document does not cover the functionality of certain methods such as getters, setters, and constructors that can be understood from reading the code. The goal of this document is to provide explanations and caveats of each class, methods, and tests.

Battleship is a 2-player board game. To start the game, each player places 5 ships (battleship, destroyer, minesweeper, and submarine) on their board. Each player then has the opportunity to end the game, move their fleet (north, east, south, or west), attack their opponent, or use a perk. The game ends when one of the players decides to end the game on their turn, or a player sinks all of their opponent's ships.

We used Java 11, Junit 5, and Lanterna 3 for this project. JUnit is a Java unit testing library. All of our testing suites instantiate objects and variables before each test with a method that uses the @BeforeEach annotation. Each unit test is marked with the @Test annotation. For JUnit, our main resource for information was the official documentation. The JUnit documentation did not need to be referred to too much, because the testing suites for each class is not very complicated. The main issue we ran into was with Lanterna. The documentation for the library is not very comprehensive, and provides many incomplete documentation. Majority of the future refactorings that will need to be made in the game class will be refactoring the GUI functionality to remove repeated code.

Development Process

We agreed to build our project using an Agile (Extreme Programming) and Kanban hybrid methodology. We planned on doing two week sprints. We have laid out 5 columns in our github:

- Brainstorming/Backlog
- To Do
- In Progress
- Testing
- Done

This allowed us to better organize what we need to accomplish during a sprint. Our backlog column holds all tasks that we know that we will have to do over the course of the project. The To do column is for tasks that will be done during the current sprint. The In Progress column is for things that are currently being worked on. The Testing column consists of tasks that need to be tested by other team members so ensure each aspect is working correctly. Last, the Done column is for tasks that have been completed. We cleared the Done column after each sprint to stay organized. We will also assign ourselves to a task, so that github will tell other team members what team member is working on a specific task. This hybrid approach allowed us to incorporate various design methodologies to achieve the best of what agile, continuous scrum, and Kanban offer.

Requirements & Specifications

Technology Stack

We are utilizing the following technologies for our project:

- git
- Slack
- Github
- Java 11
- Laterna
- IntelliJ IDEA IDE
- google-java-format IntelliJ plugin

Coding Standards

Our group has agreed to adhere to and apply the Google Java Style Guide. We feel this is a succinct and concise java style guide and adhering to it serves to benefit us should we later decide to apply for jobs at Google. Additionally, we chose Google's style guide because they are a reputable tech company with products that have scaled well, which means they have a well maintained code base. Google has made the documentation for their style guide public and it can be found at the following link: https://google.github.io/styleguide/javaguide.html.

User Stories

- Ensure that code is well tested, to mitigate bug risks as the project scales.
- Ensure our system does not violate any OOAD design principles.
- As the system grows, ensure we have high cohesion and loose coupling.
- Ensure new features added to the system are open to extension but closed to modification.

Core Functionality

- Interactive terminal based UI
- Multi-player game (autonomous agent, or human player)
- Ship placement on a board
- Players have access to perks at certain stages of the game
- The game should end once a player has sunk all opposing players' ships.

Project Risks

- One of biggest project risks during the project was connecting the backend to our frontend implementation. This was outside our wheelhouse as a group, but we managed to learn enough Laterna to get through implementing a basic game UI.
- Another large risk we encountered throughout the project was keeping up with SOLID principles. However, every sprint we agreed on a particular part of the project that was primed for refactoring and targeted these portions for the code for refactoring opportunities.

Architecture and Design

Constants Class

The Constants class was defined to maintain code readability, and to prevent accidental usages of certain values. The variables defined in Constants are used in Game, Player, Perks, Janitor, Ship, Submarine, Destroyer, Minesweeper, Battleship, and Board.

Ship Class

We defined the Ship class with the intention of leaving it as an abstract parent class from which all other ship sub classes could inherit their basic methods and member variables, but we later decided to allow for instantiation of the parent class itself to allow for refactoring to the case in which we would want an arbitrary instance of a ship that could be dynamically restructured during gameplay. The basic functionality of all ships is defined in this class.

Battleship Class

The Battleship class inherits all methods from the parent ship class. It is initialized with:

Health: 4
Captain's Quarters: index 2

Destroyer Class

The destroyer class inherits all methods from the parent ship class. It is initialized with:

Health: 3
Captain's Quarters: index 1

Minesweeper Class

The minesweeper class inherits all methods from the parent ship class, but it overrides the captainsQuartersHit() function to handle hits to its Captains Quarters differently than the other ships as specified below. It is initialized with:

Health: 2
Captain's Quarters: index 0

Submarine Class

The Destroyer class inherits all methods from the parent ship class. The submarine has a different shape than the other ships, however since it does not need to know its own location, this is accounted for by the board class which tracks the locations of every index of every ship. It is initialized with:

Health: 5
Captain's Quarters: index 3

Class and Method Summaries

Ship Class

The ship class defined the data and functionality that all ships should have. It has methods that allow for the initialization of ship health, the updating of ship health, and initialization of a captain's quarters.

String typeName: this is the name of the ship
int health: this is the health of a ship, and is updated when a section of the ship is hit
int [ ][ ] hits:

Variable Data Type and Name	Description
String typeName	This is the name of the ship
int health	This is the health of a ship, and is updated when a section of the ship is hit
int [ ][ ] hits	The first column of this variable has a 1 in the spot where there is a captain's quarters, and 0 where there is not a captain's quarters. The second column of this variable contains the health of each spot of the ship.

Method Name and Description	Corresponding Test and Description
updateHealth(int update) ● Sets a new health value for ship	canUpdateHealth() ● Initializes a ship object, and then checks if the return value from updateHealth matches what the ship was initialized with
initializeCorrectHealth(int startHealth) ● Sets the health of each spot of a ship to 1 ● Used to initialize ship health at game start	canSetHealth() ● Initializes a ship object, and then checks if the return value from updateHealth matches what the ship was initialized with
updateHits(int i, int j, int a) ● Helper function to handle MS captains quarters	canUpdateHits() ● Sets a non-captain's quarters spot in the ship to a 1, and then check that the

	spot was actually updated to a 1
takeDamage(int shipIndex) ● Deal damage to ship	canTakeDamage() ● Applies damage to a non-captain's quarter section of a ship, and checks that the spot is a 0
setCaptainsQuarters() ● Initialize captains quarters at game start	hasCaptainsQuarters ● Sets the captain's quarter of a ship to index 1. This does not check if the captain's quarter was actually set correctly

captainsQuartersHit(int index)
● Called from takeDamage, handle hit to CQ
	captainsQuartersHit()
	● Check that hitting a captain's quarters will sink a ship
	checkCaptainsQuarterError()
	● Checks that -1 is returned when no captain's quarters exists on a ship, but is getCaptainsQuarters() is called

captainsQuartersHit(int index)	Override parent definition to destroy the entire ship on CQ hit.

Player Class

Variable Data Type and Name	**Description**
String name	Name of player
Int score	Player's score
HashMap<Point, Integer> record	Key is the location on the opposing player's board that the player has tried to hit. Value is what was found there at the location (sea, ship, sub, or stacked ship and sub)
Board b	The player's board
Perks p	The player's perks
Boolean activationCode	True when the player has the access code for the laser. False otherwise
Int countHits	Store the number of hits the player has made

	against the opposing player, so that certain perks can be activated when countHits reaches a threshold.
Boolean nuke	True when the player has access to the nuke. False otherwise.
HashMap<Point, Integer> revealed	Saves locations on opponent's board that have been revealed by the sonar
Boolean sonar	True when the player has access to the sonar. False otherwise
Int bomberAvail	True when the player has access to the bomber. False otherwise

Method Name and Description	**Corresponding Test and Description**
printRecord() ● Prints player record	There is currently no test for this method
setActivationCode() ● Sets activation codes for perks	There is currently no test for this method.
moveFleetPlayer(char direction) ● Player method for activating move fleet perk	There is currently no test for this method. It is indirectly tested through BoardTest
placeShip(int x1, int y1, int x2, int y2, int health, String ship) ● Places player ships on the board	testPlaceShip() ● Places a horizontal ship, and hits a non-captain's quarters spot, and captain's quarters spot ● Places a vertical ship, and hits a non-captain's quarters spot, and captain's quarters spot ● Tries to place a ship diagonally, and checks that an error value was returned ● Tries to place a ship where the length of the ship doesn't match the distance between the entered coordinates
hit(int x, int y, Player enemy, Boolean code) ● Determines player hits and misses	testHit() ● Places a non-submarine ship, and submarine. Checks that after hitting certain spots on the opposing player's board, with a bomb, that the correct values are returned based on what was

	located on the opponent's board. testLaserBoth33() ● Test hitting a stacked non-submarine and submarine with a laser on captain's quarters (captain's quarters of both ships are in the same location) testLaserBoth32() ● Test hitting a stacked non-submarine and submarine with a laser where there is a captain's quarters for the submarine, but just a regular spot for the non-submarine testLaserSub() ● Sink a non-captain's quarters section of a submarine with a laser testLaserShip() ● Sink a non-captain's quarters section of a non-submarine ship with the laser
b2Bomber(Player enemy) ● Allows player to use b2Bomber perk	There is currently no test for this method.
lookupRecord(int x, int y) ● Looks up a player's record	testLookupRecord() ● Adds a value to a player's record, and checks if the value was added to record correctly ● Tries to access a location in record that doesn't exist. Should get an error value
addRecord(int x, int y, int hitMiss) ● Adds a location where the player tried to hit their opponent, and whether or not they hit their opponent's ship(s)	testAddRecord() ● Checks that a record can only be added once. ● There is currently no test to check an update to a record

Game Class

Variable Data Type and Name	**Description**
Player playerA	Instance of the player class to instantiate for player one when running the game.
Player playerB	Instance of the player class to instantiate for player two when running the game.
Boolean whoseTurn	A Boolean value that switches each time a turn ends. Used to indicate which players turn it is, initialized to Constants.PlayerA
Terminal terminal	Imported from Lanterna package, lowest layer of our GUI passed in to initialize Screen object. See further documentation here: https://github.com/mabe02/lanterna/blob/master/docs/using-terminal.md
Screen screen	Fundamental Lanterna object on which the terminal is instantiated. Serves as the foundation of our entire GUI. See further documentation here: http://mabe02.github.io/lanterna/apidocs/3.0/com/googlecode/lanterna/screen/Screen.html
WindowBasedTestGUI textGUI	Another step in the endless recursive instantiation of classes needed to bring up a simple gui. This one needs to take window class as an argument. See further documentation here: http://mabe02.github.io/lanterna/apidocs/3.0/com/googlecode/lanterna/gui2/WindowBasedTextGUI.html

Method Name and Description	**Corresponding Test and Description**
runGame(Scanner readIn) ● Instantiates and runs a game ● Called by the Game constructor	testGame() ● Enters sample input for an entire game. ● This test doesn't test all possible

	inputs, but does provide 100% line coverage.
runTurn(Player whichPlayer, Scanner readIn, Player enemy ● Called by runGame()	testGame() ● Indirectly tested through testGame(), which creates an instance of a Game
boardSetup(Player p, Scanner readIn) ● Sets up the board for a game	testGame() ● Indirectly tested through testGame(), which creates an instance of a Game
userMoveFleet(Player p, Scanner readIn) ● Directs player to move fleet in UI	testGame() ● Indirectly tested through testGame(), which creates an instance of a Game
userAttackOpponent(Player whichPlayer, Scanner readIn, Player enemy) ● Attacks opponent during game	testGame() ● Indirectly tested through testGame(), which creates an instance of a Game
compare(int hitStat, int record) ● Determines ships hits	testGame() ● Indirectly tested through testGame(), which creates an instance of a Game
dealPerks(Player currentPlayer, Scanner readIn, Player enemy, int correctInput) ● Directs players to use perks via the UI	testGame() ● Indirectly tested through testGame(), which creates an instance of a Game
guiSuperPrintRecord(HashMap<Point, Integer> record, HashMap<Point, Integer> rev) ● Display hits and misses from opponents board via the the Laterna UI	● There are no tests for the GUI as there is no way to test the GUI through code. ● Testing for the GUI was done via manual testing.
superPrintRecord(HashMap<Point, Integer> record, HashMap<Point, Integer> rev) ● Display hits and misses from opponents board	● There are no tests for the GUI as there is no way to test the GUI through code. ● Testing for the GUI was done via manual testing.
readIn() ● Reads player inputs from the UI	● There are no tests for the GUI as there is no way to test the GUI through code. ● Testing for the GUI was done via manual testing.
birthWindow(String info, String title, int col, int row) ● Generates Laterna window/player UI for players	● There are no tests for the GUI as there is no way to test the GUI through code. ● Testing for the GUI was done via manual testing.
birthWindowWithClose(String info, String title) ● Same as previous but with close button	● There are no tests for the GUI as there is no way to test the GUI through code.

	• Testing for the GUI was done via manual testing.

Perks Class

Variable Data Type and Name	**Description**
Janitor j	Janitor class instance that is instantiated for the purpose of cleaning the board and the records of each player after perk actions.

Method Name and Description	**Corresponding Test and Description**
sonar(Point coord, Board b) • Activates sonar perk, shows parts of the map	testSonar() • Places a sonar at (3, 3), and checks that sonar() returns a HashMap with the correct key-value pairs
moveFleet(Board b, char direction) • Allows player to move their entire ship fleet	testMoveFleet() • Moves a set of ships, and checks that the ships that were able to be moved in the inputted direction were moved.
undoMove(char move, Board b) • Allows a player to undo their most recent move	testUndoMove() • Places a set of ships on the board and moves the fleet in a certain direction. Calls undoMove(), and checks that ships are still in their original locations.
redoMove(char move, Board b) • Allows a player to redo their previous move.	testRedoMove() • Places a set of ships and moves the fleet. The test then checks that an undoMove() with a redoMove() right after results in the ships being in the same locations as they were after the fleet was moved.

Board Class

Variable Data Type and Name	**Description**

Int[10][10] board	A 2-D array for which each index has a corresponding value that indicates what is at that space on the board.
HashMap<Point, ArrayList<Ship>> shipLocations	A hashmap that contains the locations of every ship object. The key is the coordinate pair represented as a Point and the value is an ArrayList of Ship objects that allows for overlap of a ship and a submarine.
MineSweeper ms	An instance of MineSweeper class.
Destroyer ds	An instance of Destroyer class.
Battleship bs	An instance of Battleship class.
Submarine ss	An instance of Submarine class.
ArrayList<Point> msOrientation	ArrayList of Points that hold the coordinates of the MineSweeper and that maps it to its representation in the Ship class.
ArrayList<Point> dsOrientation	ArrayList of Points that hold the coordinates of the Destroyer and that maps it to its representation in the Ship class.
ArrayList<Point> bsOrientation	ArrayList of Points that hold the coordinates of the Battleship and that maps it to its representation in the Ship class.
ArrayList<Point> ssOrientation	ArrayList of Points that hold the coordinates of the Submarine and that maps it to its representation in the Ship class.
HashMap<String, ArrayList<Point>> masterOrientation	HashMap that contains the name of the ship as a string and its respective orientation ArrayList as its value.
String message	String where messages pertaining error checking of the Board are stored for the UI.

Method Name and Description	**Corresponding Test and Description**
printBoard()	Prints board after every player turn.
bombApplyDamage(int x, int y)	Applies bomb damage to ship when hit

laserApplyDamage(int x, int y)	Applies laser damage to ship when hit
lengthCheck(int x1, int y1, int x2, int y2, int length)	Input Validation. Checks that coordinates match the ship's health.
diagonalBoundsCheck(int x1, int y1, int x2, int y2)	Checks if a ship was placed diagonally.
outOfBoundsCheck(int x1, int y1, int x2, int y2, String shipName)	Checks if a ship was placed out of bounds.
setShip(int x1, int y1, int x2, int y2, int health, String ship)	Places a ship on the board, as well as saving its location in the shipLocations HashMap and in its corresponding orientation ArrayList.
setSub(int x1, int y1, int x2, int y2, int health, String ship)	Places a submarine on the board, as well as saving its location in the shipLocations HashMap and in its corresponding orientation ArrayList.
setShipLocations(Point coord, String ship)	Updates the shipLocations HashMap.
setShipArray(int x1, int y1, int x2, int y2, String ship)	Updates the corresponding orientation array for a ship.

Janitor Class

Method Name and Description	**Corresponding Test and Description**
cleanupOnAisle5(Board b, Ship s, ArrayList<Point> positions, String ship) ● Cleans a sunken ship from the board and its associated data structures.	testCleanupOnAisle5() ● Places ships on board, and checks that calls of cleanupOnAisle5 on each ship correctly removes all ships from the board.

Major Refactorings

Player Class

Early on we made the mistake of trying to have the player classes maintain their own instances of a board. This idea seemed intuitive at first because in the battleship board game the players have their own boards. We later realized that this was in violation of SOLID as well as OOAD principles. We learned during lecture that this implementation would lead to our code becoming poorly cohesive and less extendable. As a result, we decided to add the board class to our code. Adding the board class indeed made our code much more extensible and cohesive. In hindsight we are glad we caught this and refactored it early on. Had we not we would have ran into some serious issues maintaining our project throughout the semester.

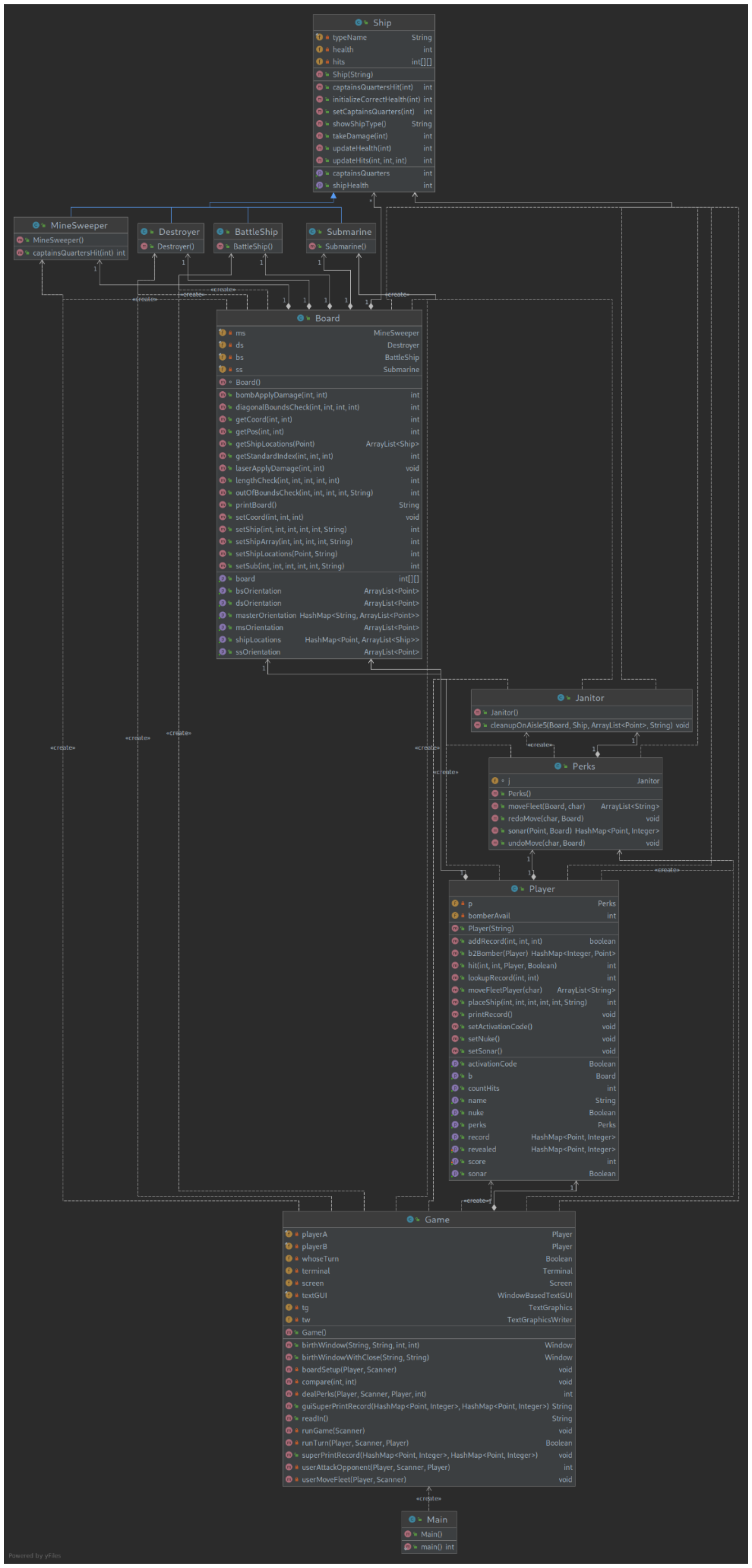
Board Class - getCoord( ) method

One of our larger hurdles early on in the project was the getCoord() method of our board class. In our first iterations the getCoord() quickly became a bloated method with too much responsibility. We quickly realized that this was a violation of the SOLID principles because we had a single method in one class controlling too much responsibility. We broke up this method into several new methods. After that, we placed the new methods in the appropriate classes where necessary and even created new classes when necessary. As a result, we were able to refactor a method that was turning our board class into a god class. The refactoring made our code in line with the SOLID principles as well as made our code more extensible and cohesive.

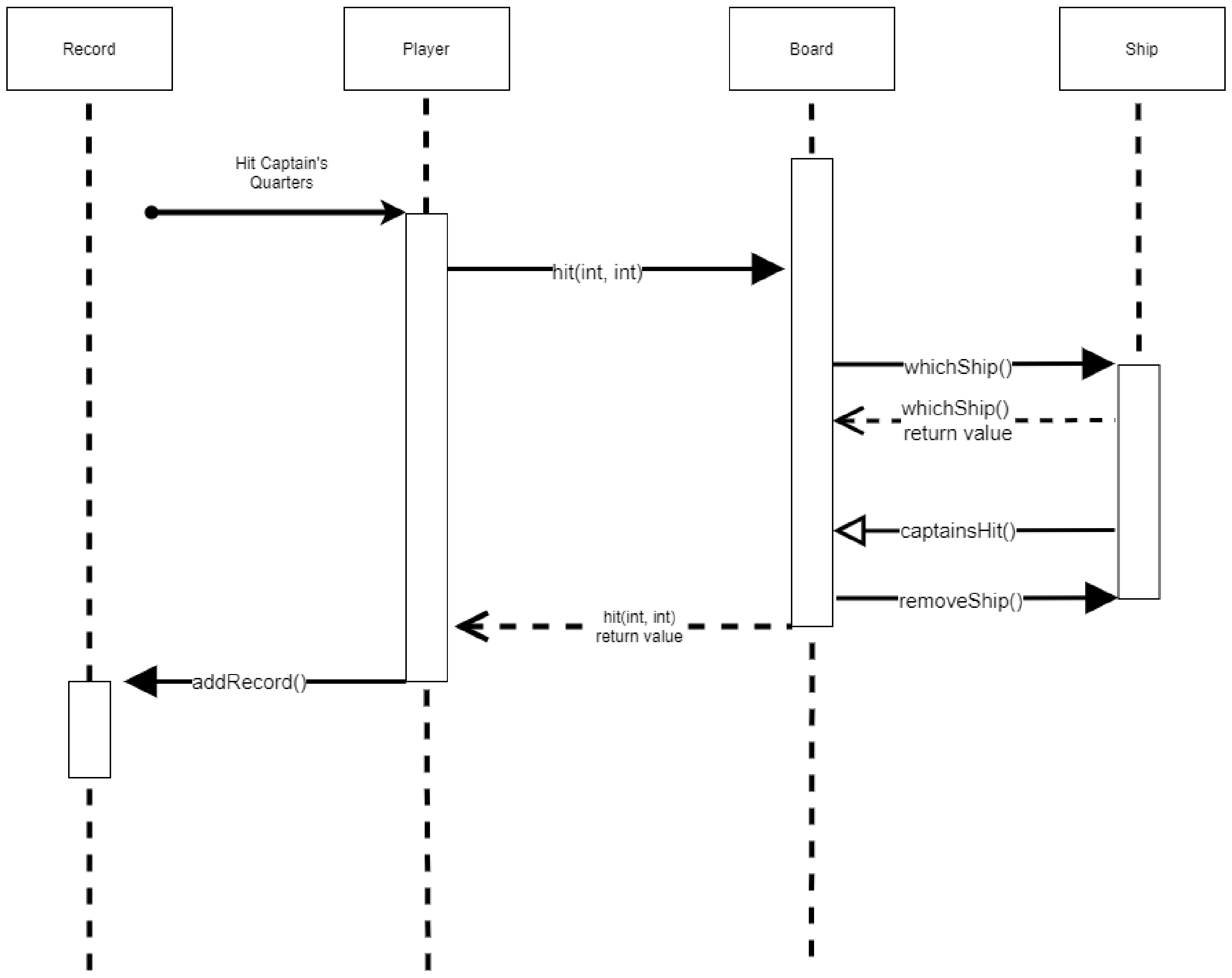
Frontend UI - Switching from JavaFX to Laterna

Initially, we decided to use JavaFX and SceneBuilder, which would have allowed us to implement a UI. We had a hard time figuring out how to switch between players. This is a very basic operation in the game. Trying to implement such a basic part of the game came with a very steep learning curve for our group. There was too much to learn for game development in JavaFX. As a result, we decided to move to find a library that would allow us to keep the majority of our Game class functionality, but that would also allow us to have a more user-friendly way of playing the game. We ended up choosing Lanterna as it had the most up to date documentation. Although the documentation itself is of poor quality, it also seemed to be the easiest to integrate with our existing code. This decision was made in order to complete the game before the due date of the project.

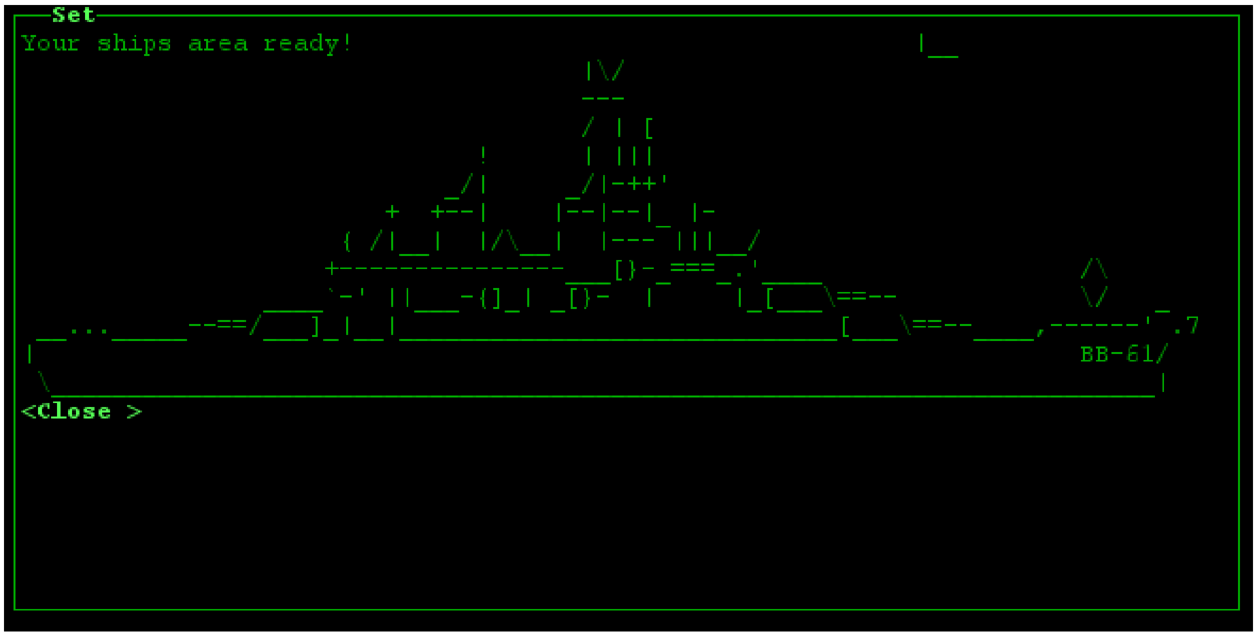
UML Class Diagram


Ship
typeName String
health int
hits int[][]
Ship(String)
captainsQuartersHit(int) int
initializeCorrectHealth(int) int
setCaptainsQuarters(int) int
showShipType() String
takeDamage(int) int
updateHealth(int) int
updateHits(int, int, int) int
captainsQuarters int
shipHealth int
MineSweeper
MineSweeper()
captainsQuartersHit(int) int
Destroyer
Destroyer()
BattleShip
BattleShip()
Submarine
Submarine()
«create»
Board
ms MineSweeper
ds Destroyer
bs BattleShip
ss Submarine
Board()
bombApplyDamage(int, int) int
diagonalBoundsCheck(int, int, int, int) int
getCoord(int, int) int
getPos(int, int) int
getShipLocations(Point) ArrayList<Ship>
getStandardIndex(int, int, int) int
laserApplyDamage(int, int) void
lengthCheck(int, int, int, int, int) int
outOfBoundsCheck(int, int, int, int, String) int
printBoard() String
setCoord(int, int, int) void
setShip(int, int, int, int, int, String) int
setShipArray(int, int, int, int, String) int
setShipLocations(Point, String) int
setSub(int, int, int, int, int, String) int
board int[][]
bsOrientation ArrayList<Point>
dsOrientation ArrayList<Point>
masmasterOrientation HashMap<String, ArrayList<Point>>
msOrientation ArrayList<Point>
shipLocations HashMap<Point, ArrayList<Ship>>
ssOrientation ArrayList<Point>
Janitor
Janitor()
cleanupOnAisle5(Board, Ship, ArrayList<Point>, String) void
Perks
j Janitor
Perks()
moveFleet(Board, char) ArrayList<String>
redoMove(char, Board) void
sonar(Point, Board) HashMap<Point, Integer>
undoMove(char, Board) void
Player
p Perks
bomberAvail int
Player(String)
addRecord(int, int, int) boolean
b2Bomber(Player) HashMap<Integer, Point>
hit(int, int, Player, Boolean) int
lookupRecord(int, int) int
moveFleetPlayer(char) ArrayList<String>
placeShip(int, int, int, int, int, String) int
printRecord() void
setActivationCode() void
setNuke() void
setSonar() void
activationCode Boolean
b Board
countHits int
name String
nuke Boolean
perks Perks
record HashMap<Point, Integer>
revealed HashMap<Point, Integer>
score int
sonar Boolean
Game
playerA Player
playerB Player
whoseTurn Boolean
terminal Terminal
screen Screen
textGUI WindowBasedTextGUI
tg TextGraphics
tw TextGraphicsWriter
Game()
birthWindow(String, String, int, int) Window
birthWindowWithClose(String, String) Window
boardSetup(Player, Scanner) void
compare(int, int) void
dealPerks(Player, Scanner, Player, int) int
guiSuperPrintRecord(HashMap<Point, Integer>, HashMap<Point, Integer>) String
readIn() String
runGame(Scanner) void
runTurn(Player, Scanner, Player) Boolean
superPrintRecord(HashMap<Point, Integer>, HashMap<Point, Integer>) void
userAttackOpponent(Player, Scanner, Player) int
userMoveFleet(Player, Scanner) void
Main
Main()
main() int
Powered by yFiles

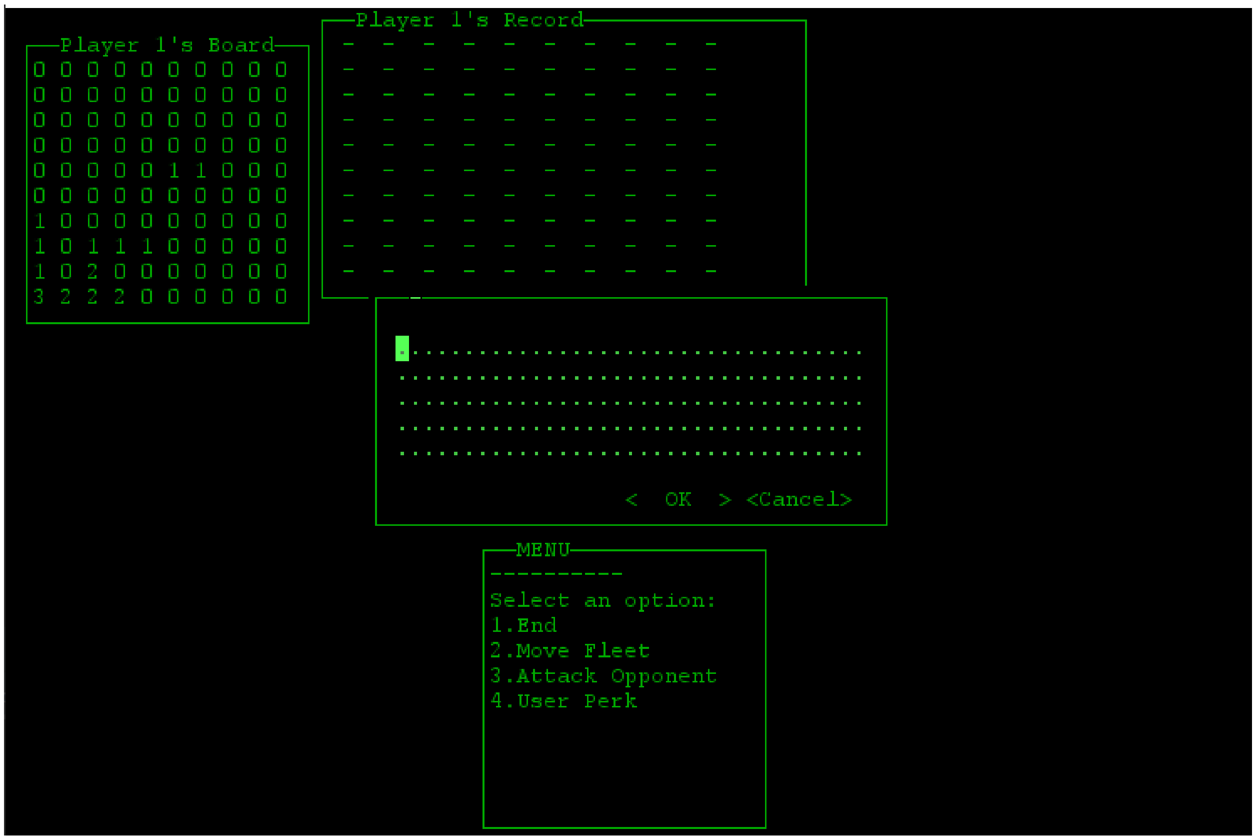
Sequence Diagram



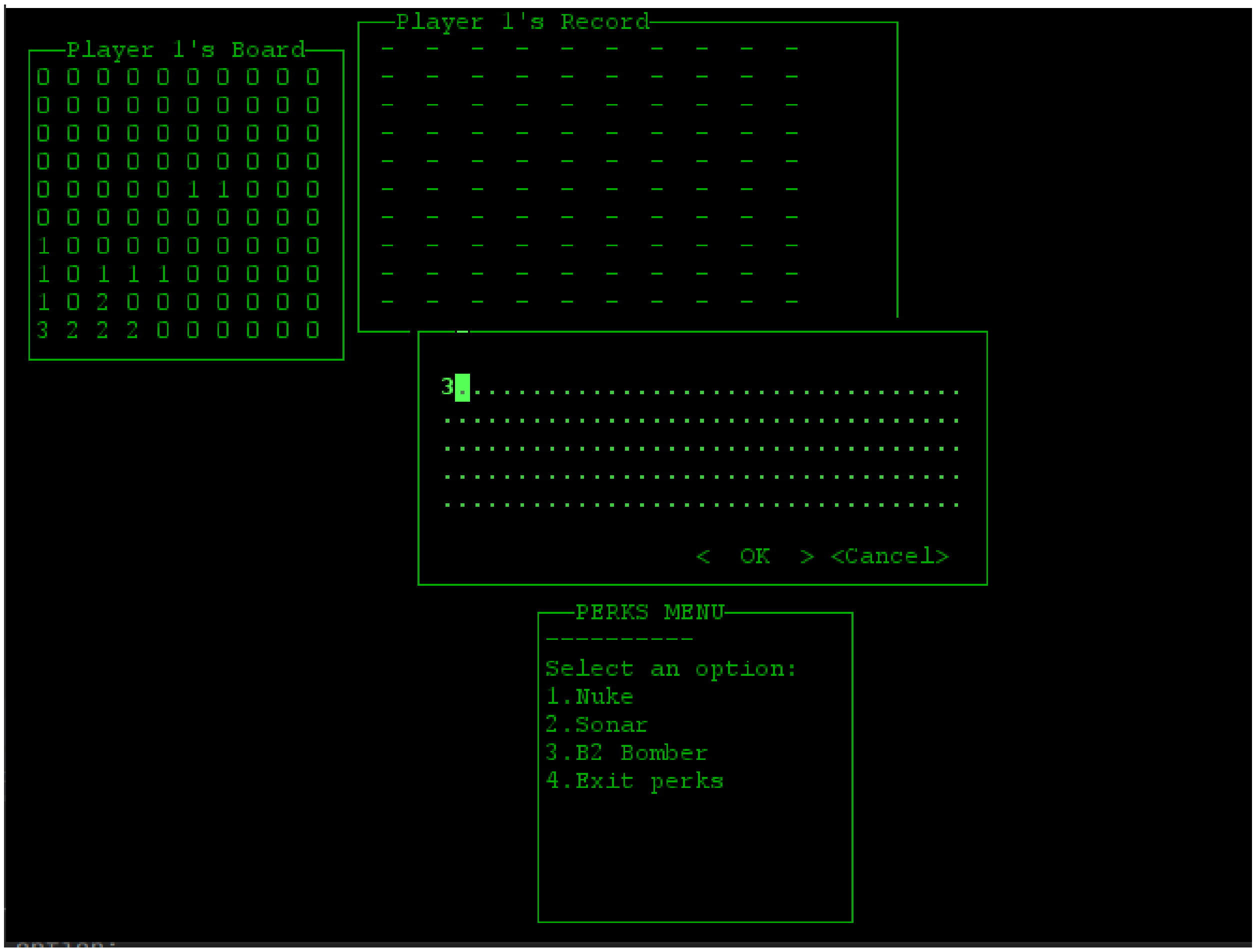
Gallery



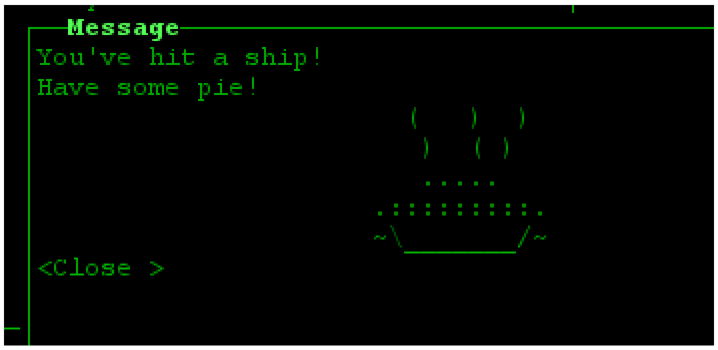
```
Set
Your ships area ready!                                      |__
                                      |\/
                                     ---
                                     / | [
                               !      | |||
                            _/|     _/|-++'
                        +  +--|     |--|--|_ |-
                     { /|__|  |/\__|  |---  |||__/
                    +----------------____[}-_===_.'____          /\
                ____ `-'  ||___-{]_| _[}-   |     |_[___\==--     \/
 __...______--==/___]_|__|______________________________[___\==--____,-------'_.7
|                                                                      BB-61/
 \                                                                          |
  ___________________________________________________________________________
<Close >
```



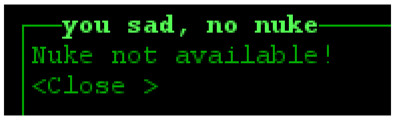
```
Player 1's Board          Player 1's Record
0 0 0 0 0 0 0 0 0 0       - - - - - - - - - -
0 0 0 0 0 0 0 0 0 0       - - - - - - - - - -
0 0 0 0 0 0 0 0 0 0       - - - - - - - - - -
0 0 0 0 0 0 0 0 0 0       - - - - - - - - - -
0 0 0 0 0 1 1 0 0 0       - - - - - - - - - -
0 0 0 0 0 0 0 0 0 0       - - - - - - - - - -
1 0 0 0 0 0 0 0 0 0       - - - - - - - - - -
1 0 1 1 1 0 0 0 0 0       - - - - - - - - - -
1 0 2 0 0 0 0 0 0 0       - - - - - - - - - -
3 2 2 2 0 0 0 0 0 0       - - - - - - - - - -

                          ...................................
                          ...................................
                          ...................................
                          ...................................
                          ...................................

                                             <  OK  > <Cancel>

                              MENU
                              ----------
                              Select an option:
                              1.End
                              2.Move Fleet
                              3.Attack Opponent
                              4.User Perk
```



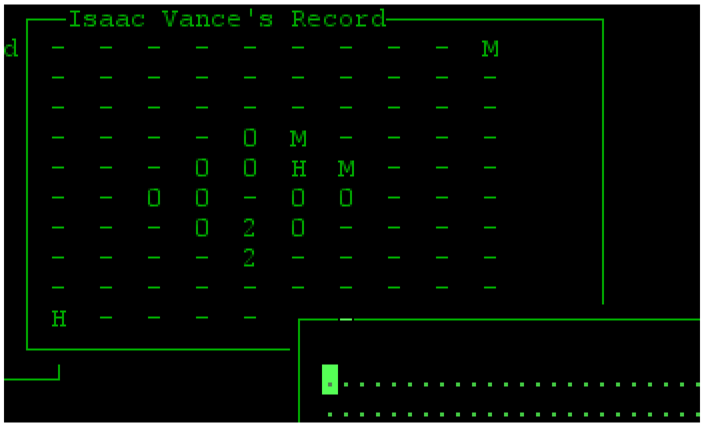
```
Player 1's Board                 Player 1's Record
0 0 0 0 0 0 0 0 0 0              - - - - - - - - - -
0 0 0 0 0 0 0 0 0 0              - - - - - - - - - -
0 0 0 0 0 0 0 0 0 0              - - - - - - - - - -
0 0 0 0 0 0 0 0 0 0              - - - - - - - - - -
0 0 0 0 0 1 1 0 0 0              - - - - - - - - - -
0 0 0 0 0 0 0 0 0 0              - - - - - - - - - -
1 0 0 0 0 0 0 0 0 0              - - - - - - - - - -
1 0 1 1 1 0 0 0 0 0              - - - - - - - - - -
1 0 2 0 0 0 0 0 0 0              - - - - - - - - - -
3 2 2 2 0 0 0 0 0 0              - - - - - - - - - -

3...................................
....................................
....................................
....................................
....................................

                  <  OK  > <Cancel>

PERKS MENU
----------
Select an option:
1.Nuke
2.Sonar
3.B2 Bomber
4.Exit perks
```



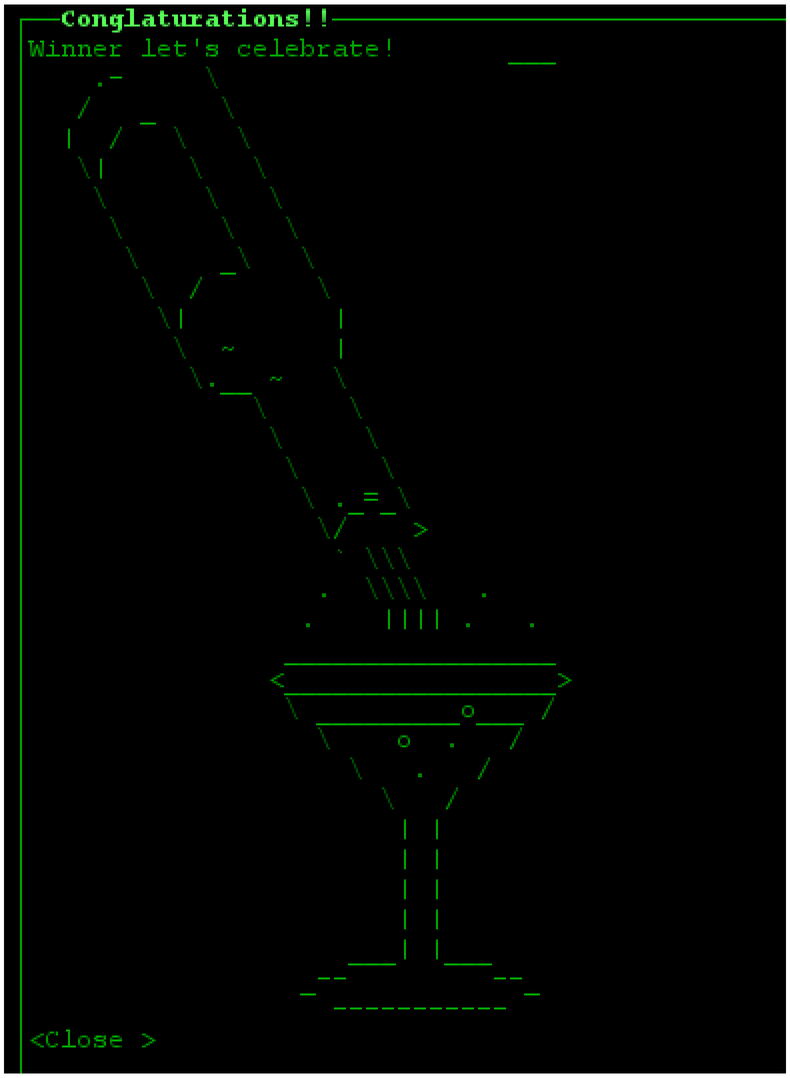
```
Message
You've hit a ship!
Have some pie!
                          (   )  )
                           )   ( )
                           .....
                        .:::::::::.
                        ~\_______/~

<Close >
```



```
you sad, no nuke
Nuke not available!
<Close >
```

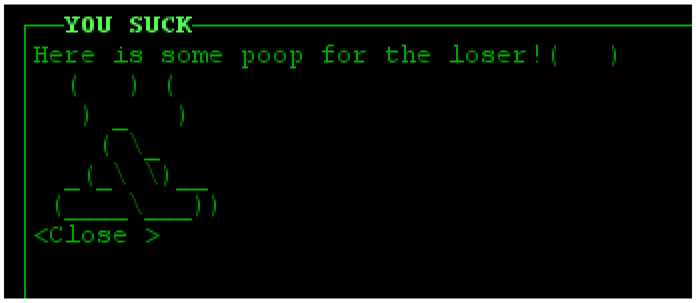


```
Isaac Vance's Record
d  -  -  -  -  -  -  -  -  -  M
   -  -  -  -  -  -  -  -  -  -
   -  -  -  -  -  -  -  -  -  -
   -  -  -  -  0  M  -  -  -  -
   -  -  -  0  0  H  M  -  -  -
   -  -  0  0  -  0  0  -  -  -
   -  -  -  0  2  0  -  -  -  -
   -  -  -  -  2  -  -  -  -  -
   -  -  -  -  -  -  -  -  -  -
   H  -  -  -  -  -
```

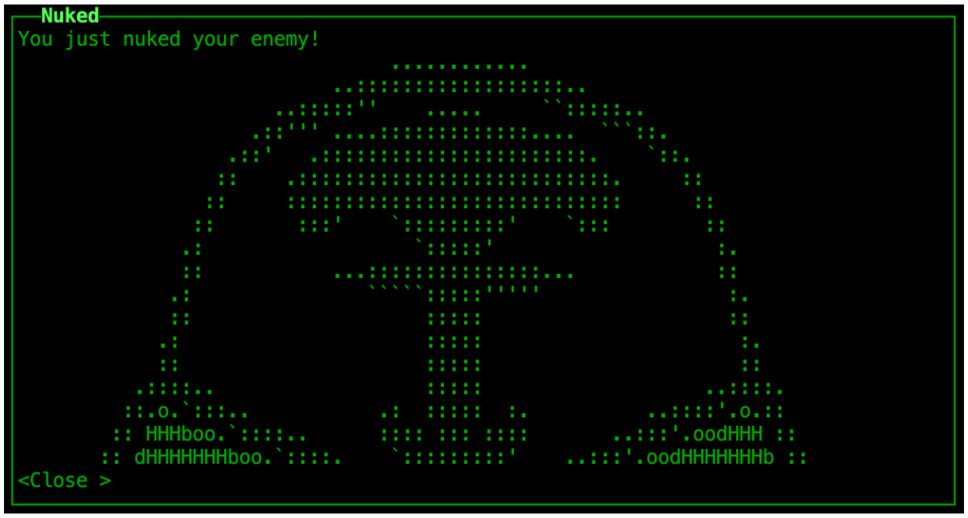


Conglaturations!!

Winner let's celebrate!



```
YOU SUCK
Here is some poop for the loser!
<Close >
```



```
Nuked
You just nuked your enemy!
<Close >
```

Personal Reflections

Isaac Vance

I began this class hoping to get hands on experience designing and implementing an Object Oriented Project, and this project did not disappoint. This group was fantastic to work with, and having committed and communicative teammates helped me immensely with the task of learning the new technologies we used for this project such as IntelliJ, the Java coding language, Maven Java framework, JUnit testing, and Lanterna Java UI. I look forward to working with this team in the future, and I am very grateful to have had this experience.

Stefan Primelles

As I began the semester I had the intention of learning Java, improving my software development skills, and learning about object oriented design, this project and my teammates were instrumental for me to achieve that. I had the opportunity to discuss at length different ideas with my team to satisfy the requirements and implement good designs. Working in this project also exposed me to JUnit and Lanterna, and made me more familiar with IntelliJ and Git. My teammates got the best out of me and together we were able to creatively come up with solutions to various problems and build a project of which we are proud of.

Jarek Carlson

My goal with this class is to be a part of a team that implements an Object Oriented Project end to end, and this project along with my team made the experience an extremely positive one. My team was fantastic. It was truly a joy to work with the other members of my group. Additionally I got to further develop my git skills with challenges I had never experienced such as advanced branching, stashing, and rebasing. I also got to learn a ton about Java and OOAD. Learning these new technologies along with my team members was a great experience and made for a truly great project experience.

Neel Karsanbhai

My expectations for this class were to learn Java, and Object Oriented Programming. Being in this team helped me learn these concepts very well. I had a great experience working with my teammates, and I am happy with how much I have learned through this experience. I ended up gaining skills through this project that I did not expect to gain, such as how to do test driven development, and how to create a JUnit 5 testing suite. These are skills that seem to be very important in industry, and I am happy knowing I can start an internship with such practical skills. Working on this project with my teammates has been a very memorable experience, and I look forward to working with them again.