Team C

Battleship Documentation

By: John Fletcher, Filip Nedelkov

Description:

Throughout the Object-Oriented Analysis/Design semester, we had to make a full scale Battleship game by using the Java skills and design patterns we learned in class. The result of our semester-long project is a simple Battleship game where two players are able to place their ships, sink their opponent's ships, and use different types of weapons to do so. There are also different types of features that are not found in typical Battleship games in order to add some uniqueness to the project. By having a three-dimensional game, multiple types of ships with all different types of layouts, and having different types of weapons, it was important to have the code's foundation easily scalable.

Development Process:

Beginning development was difficult as all four (at the time) team members were not familiar with Java. For the first two milestones, the group was still trying to figure out the best design practices to use in implementing the required features. We had to later do some refactoring of our foundation code in order to better handle scaling the game with new features and requirements. Also, we quickly learned the importance of setting up Unit Testing before implementing a feature in order to avoid any bound condition issues. We used github for our version control in order to keep track of our development process. Github proved to be very useful as we were able to independently work on features on our own time.

Requirements and Specifications:

Throughout the project, requirements and specifications changed multiple times. In order for our project to compensate for these changes, we had to continuously refactor our code to be as scalable as possible. One way to help us develop the requirements was by making user stories/user cases:

USER STORIES / USER CASES

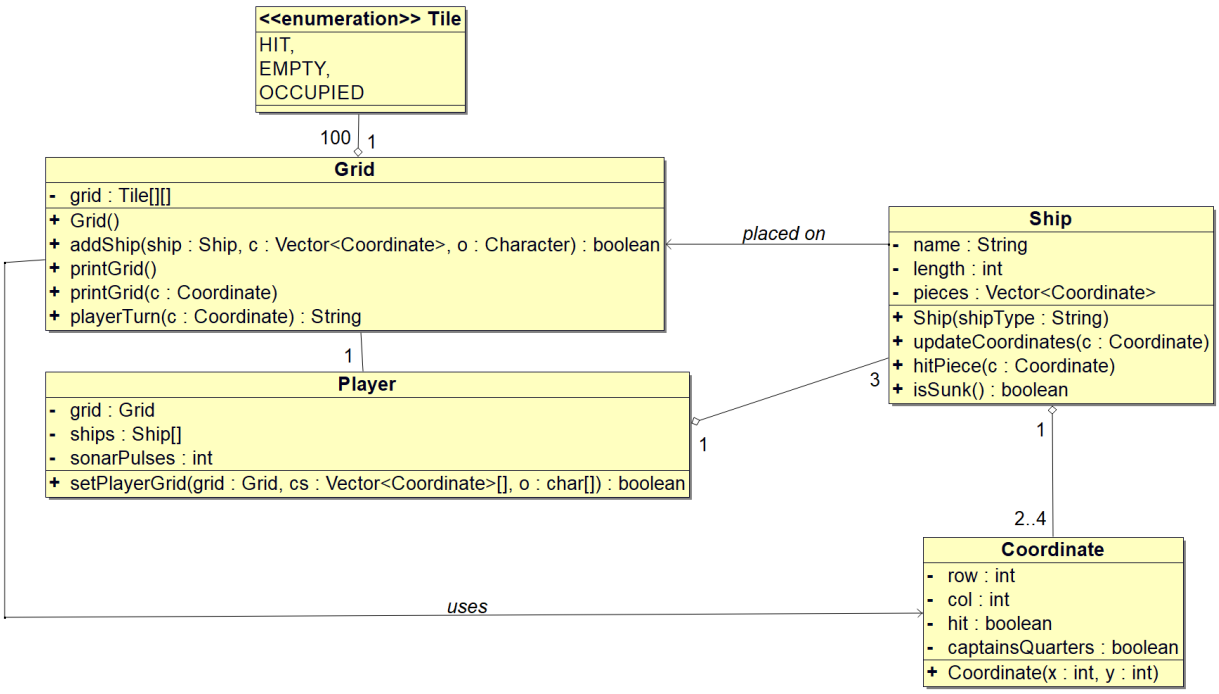
1. As a player, I want the system to keep track of where my ships are.
2. As a player, I want to be able to choose a coordinate and for the system to tell me whether I hit a ship or missed it.
3. As a player, I want the system to tell me when my ship has sunk.
4. As a player, I want to be able to use the sonar pulse in place of a turn twice in a game, once I have sunk at least one of the opponent's ships.

5. As a player, if I hit the captain’s quarters of one of the opponent’s ships, I want the whole ship to sink, unless the quarters is armored, in which case I want the ship to sink if I hit it twice.
6. As a player, I want to be able to use the space lazer once my enemy’s ship has sunk.
7. As a player, I want to be able to move my fleet of ships to any direction (N,S,E or W) and have the option to undo this move as many times as I want.
8. As a player, I want to know if I’ve won the game.
9. As a player, I want to be able to place a mine on my opponent's board before they place their ships.
10. As a player, I want to be able to continue playing even if my battleship is sunk.
11. As a player, I want to be able to place my ships using inputs on the command line.
12. As a player, I want to be able to play battleship using inputs on the command line.

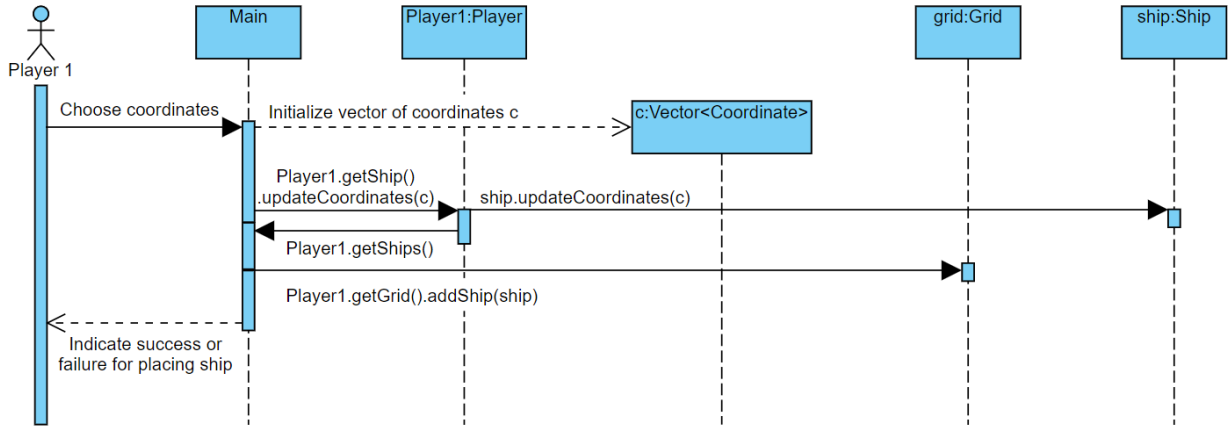
As you can see, we had to think of what a player might want to do in Battleship in order to correctly implement the requirements and specifications laid out to us in the milestones. Also, these stories are not the same as the original ones; as when the requirements changed, so did our cases.

Architecture and Design:

In order to better develop the software for the Battleship game, it was important to start the development process by outlining the structure of our code and how our classes will interact with each other through UML diagrams. Our UML diagrams below show how all the different classes interact with each other:



Another important feature we had to properly design for was how a player would add their fleet of ships to the board. To better visualize the design for placing ships, we created a sequence diagram:



For Battleship, we had to take into consideration on how a player should place ships on the grid at the beginning of the game, what happens during a turn, and how users should interact with different weapons. We created a Grid (with an enumeration of Tile), Player, Ship, and Coordinate class.

Grid:
Handles all the logic of the player's board (i.e where ships are placed, previous shot attempts, and printing the grid out)

- Enumeration of *Tile* that has 5 types
    - HIT
    - MISS
    - EMPTY
    - OCCUPIED
    - MINE
- A 10x10 array made up of *Tile* type
    - Has 2 "depth" layers

Works closely with **Tile, Player, Ship, Coordinate**

Player:
Handles all the logic of each individual player (i.e how many ships left in the players fleet, the type of weapon the player wants to use for their turn, adding a ship to the grid)

- Each player has their own Grid where they place their ships
    - On a turn, the player sees a "hidden" grid of their opponents
- A player has a vector of weapons that make up their arsenal
    - Each type of weapon can be used at different times throughout a game

Works closely with **Grid, Ship**

Ship:
Handles all the logic of how a ship should interact with both the board and the players (i.e checking if the ship is sunk, checking if a coordinate is specific to the Captains Quarters, and updating the ships coordinates when a move is requested by the player)

- Ship is an abstract class where we have different types of ships
    - Battleship (length 4)
    - Destroyer (length 3)
    - Lifeboat (length 1)
        - Gets randomly placed once the Battleship is sunk
    - Minesweeper (length 2)
    - Submarine (length 5)

Works closely with **Player, Grid, Coordinate**

Coordinate:
Defines what a coordinate is for the Grid and Ship class (i.e row and column, valid coordinate, and comparing two coordinates)

- A coordinate is made up of a row, column, and possibly a depth (to take into account the submarine)

Works closely with **Grid, Ship**

Weapon:
Handles all the logic of how weapons interact with the Player and Grid (i.e the availability of the weapon to the player, the direct specifications of a certain weapon)

- Weapon is an abstract class where we have different types of weapons
    - Sonar
        - Only available to use once in a game
        - Checks to see if there is an enemy ship within a certain distance of the coordinate
    - Space Laser
        - Unlocked when a player sinks their first ship of the game
        - Can shoot through both depth 0 and depth 1
        - Can hit two different ships at depth 0 and depth 1
    - Bomb
        - The weapon a player starts off with
        - Only hits a single Tile each turn

Works closely with **Player**

Main:
Handles the overall logic of how the Battleship game works (i.e create the game, facilitate turns in between players, and checks when the game is done (when a player ships have all been sunk))

Reflection:

John:

Overall, I really enjoyed the process of creating this Battleship game. Not only did it teach me how to properly create code with design patterns, but it also helped me become more comfortable working with new technologies and overcoming the obstacles that come with it. One thing I wish I took into consideration when losing two team members in the middle of the semester was the shift of work that needed to be done. Being down two people who were heavily involved with the design process made it difficult to keep the momentum going.

However, Filip and I quickly picked up the pace and finished a battleship game that we are both proud of.

Filip:

This project was an overall good experience. I learned a lot about design patterns and how to work well on a team. We had a great team that communicated well, and it felt like all of us wanted to contribute as much as we could to the project. One thing that could have gone better was the initial way we structured the project. We did not fully take into account how complicated adding new features would become, so it forced us to refactor a decent amount of our code. If we had taken more time at the beginning it would have made things easier in the long run. All in all, I am proud of our work on this project even though we might not have reached the amount of new features that we wanted to implement by the end.