VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

**REPORT PROJECT OF DATA STRUCTURE AND ALGORITHM**

**NAME PROJECT: BATTLESHIP**

By

NGUYEN VIET ANH – ITDSIU18027

NGUYEN NGOC HUY – ITDSIU18052

DATA STRUCTURE AND ALGORITHM

Ho Chi Minh City, Vietnam

2023 – 2024

**REPORT PROJECT OF DATA STRUCTURE AND ALGORITHM**

REMARK: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

LECTURER SIGNATURE

**TABLE OF CONTENT**

1. **Structure**

The rules of the game

Levels of the game

How the core game work

Extra feature in future

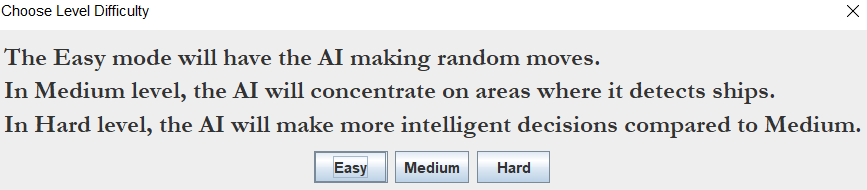
**1. The rules of the game**

Battleship is a strategic game where two players use a 10x10 grid and position 5 ships of varying lengths (5, 4, 3, 3, and 2 units) in any horizontal or vertical arrangement. The game starts with each player unaware of their adversary's ship placement. During the game, players take turns to target unattacked positions on the opponent's grid. After an attack, the position is marked: a blue marker indicates a miss in open water, whereas a red designation signifies a successful hit on a ship.

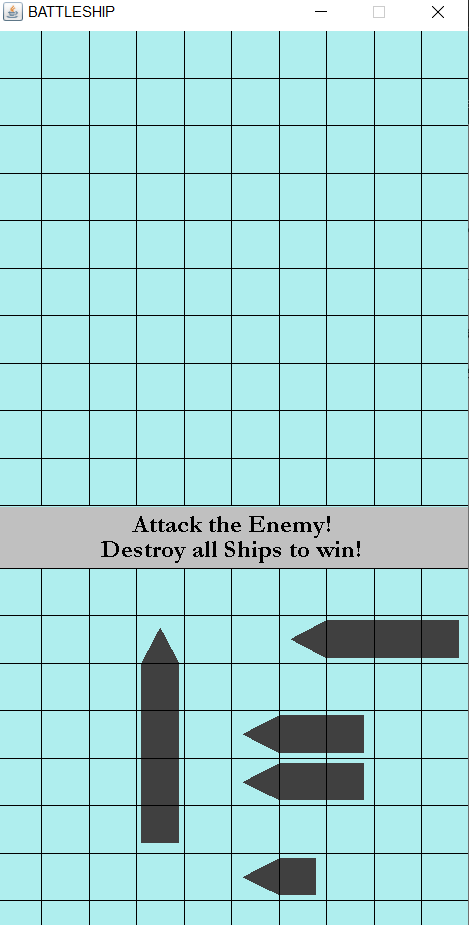
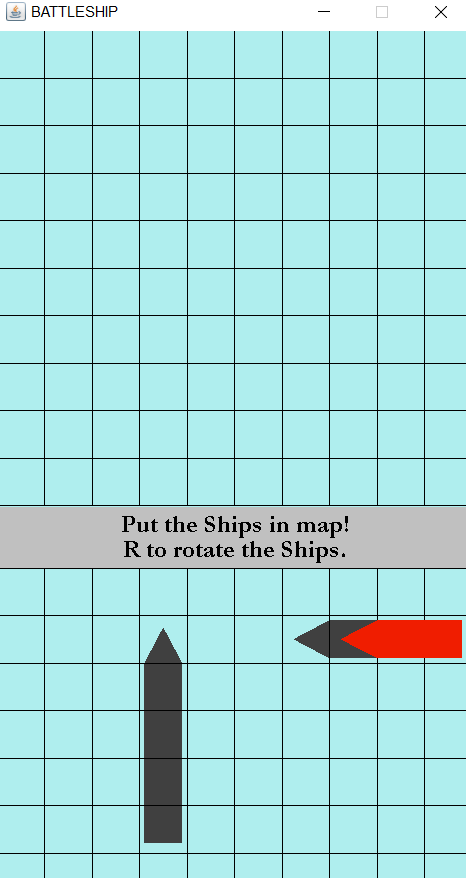
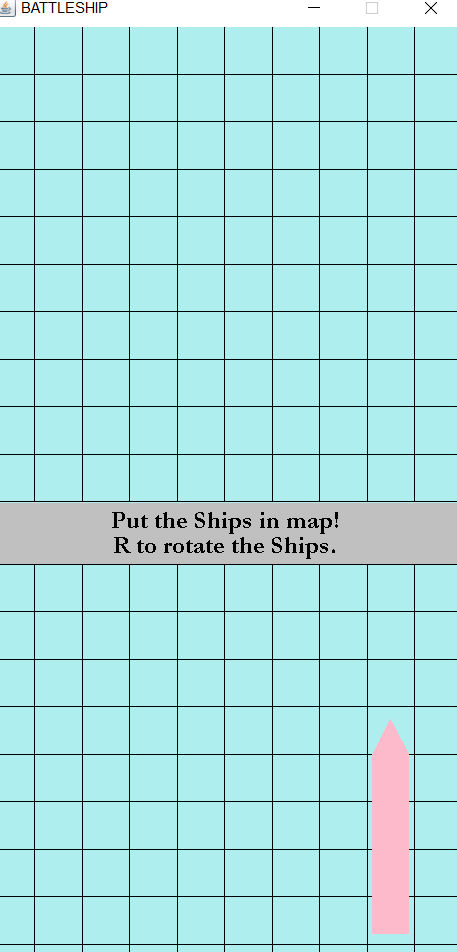
Once all grid positions of a ship are hit, the vessel is exposed as destroyed to the opponent. The game reaches its conclusion when a player's entire fleet has been obliterated. The victor is the player whose ships remain intact.

**2. Level of the game**

This example of the Battleship game has integrated various AI levels to enable single-player gameplay. Upon game start, a dialogue box is displayed before launching the main game interface, which allows the player to choose their preferred difficulty level.

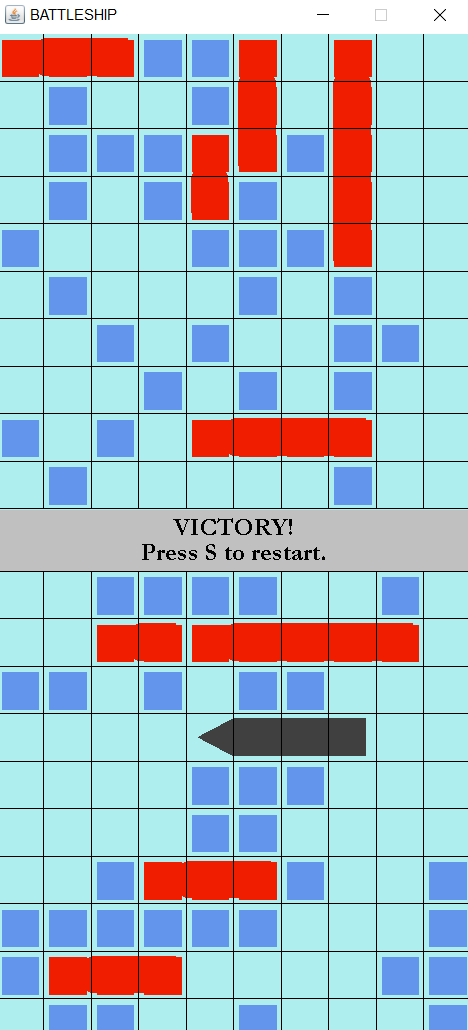
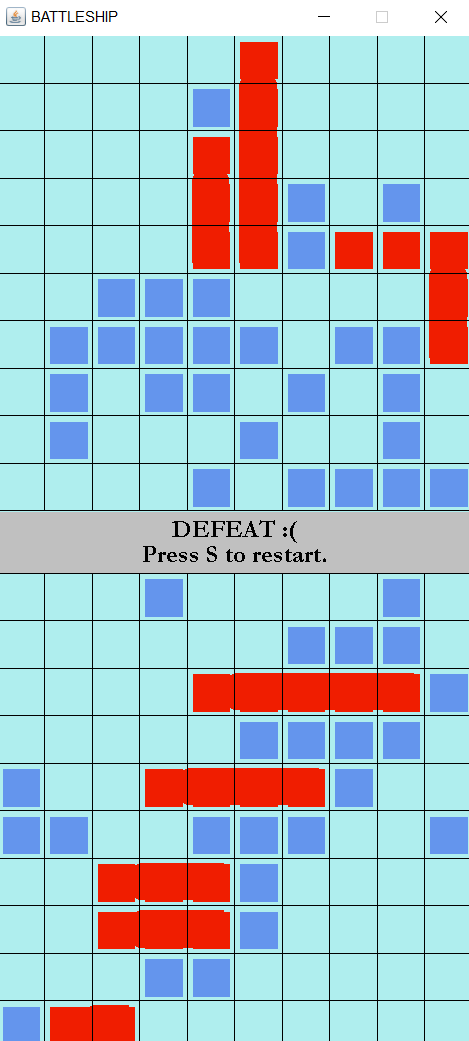
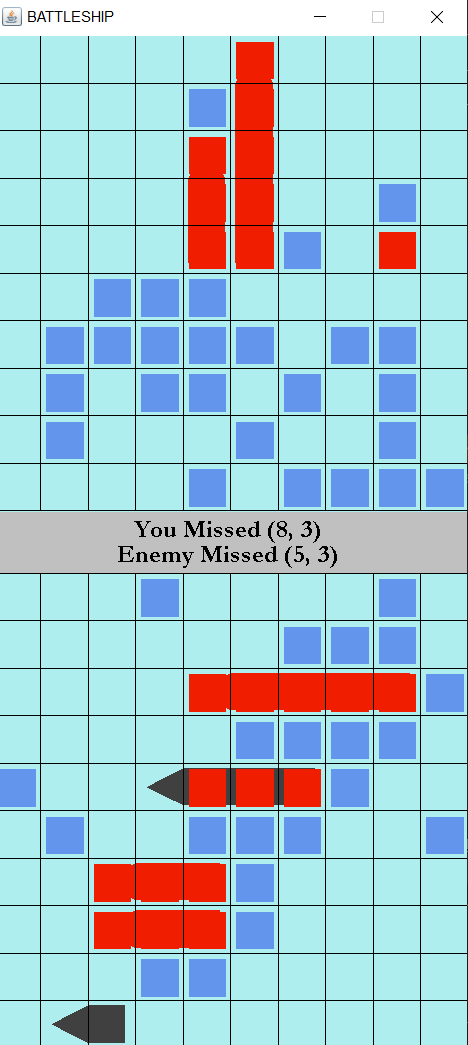


The Easy AI mode makes all decisions completely at random. In contrast, the Medium AI randomly selects cells, but once it hits an undestroyed ship, it will continue to target neighboring cells until the ship is sunk. On the other hand, the Hard AI employs a more strategic approach, factoring in the number of surrounding cells when making its choices. Upon hitting a ship, it attempts to create a line of successful hits instead of merely choosing adjacent cells randomly.



Upon launching the game, the player enters the stage where they arrange their ships. The three images above demonstrate the progression of this process. In the first image, one ship has been set, and another is poised to be positioned. The pink color denotes that the location chosen for the ship is acceptable. A left click of the mouse would secure the ship's placement. The middle image depicts the ship in red, signaling an invalid placement due to overlap with an already stationed ship. Once all ships are properly positioned, the screen shown in the last image appears, indicating that the game has transitioned to the attack phase. Players can then start making strikes on the upper grid.

Clicking on each cell will classify the cell as either a hit or a miss, represented by red or blue markers respectively. A destroyed ship is signified by the complete ship turning red beneath the red markers. The game concludes when all the ships on one side of the grid are annihilated, resulting in either the player's victory or the computer's triumph. The rightmost image below illustrates this end-game state.



The game controls can be summarized as follows:

* Universal commands:
  + Pressing 'Escape' quits the game.
  + 'S' restarts the game.
  + 'D' triggers the debug mode, allowing the player to cheat by revealing the enemy's ships.
* During the Ship Placement Phase:
  + A mouse clicks places the ship at the chosen location, given the placement is valid.
  + The 'R' key toggles the ship's orientation between vertical and horizontal for placement.
* During the Attack Phase:
  + Clicking on unmarked locations within the enemy's grid unveils the selected squares as either hits or misses.

**3. How the core game work and UML**

This section will briefly describe the purpose of each class in the structure. It will not cover in detail all the methods that are included in each method. We can view the code to browse the in-depth comments to understand how each element has been created.

**Representing the Grid:**

·  **Marker:** Represents the colored red or blue markers that show up when attacking the grid. All 10x10 of these are created at the start and drawn if they have been marked. They keep track of the ship they would represent hitting to allow the AI to get the information.

· **Position:** Used to represent a position with an x and y coordinate mostly for indicating the grid coordinates, but in many cases also to represent pixel offsets for drawing.

· **Rectangle:** Used to represent a generic rectangle with a top corner, width, and height with collision detection against a single point.

·  **SelectionGrid:** The actual grid that contains a collection of markers, and ships, and draws these with a set of lines to show where cells are. Includes appropriate methods to manage the state of the grid.

· **Ship:** Defines a ship that keeps track of where it is located on the grid and should be drawn. The ship can also track whether it has been destroyed to notify other classes when asked.

**High-level interface:**

· **Game:** Creates the JFrame to contain the GamePanel and manages a collection of the keyboard input.

·  **GamePanel:** Controls the two selection grids for the players and manages all the player interaction with their grid. Controls all the information necessary to manage the game state.

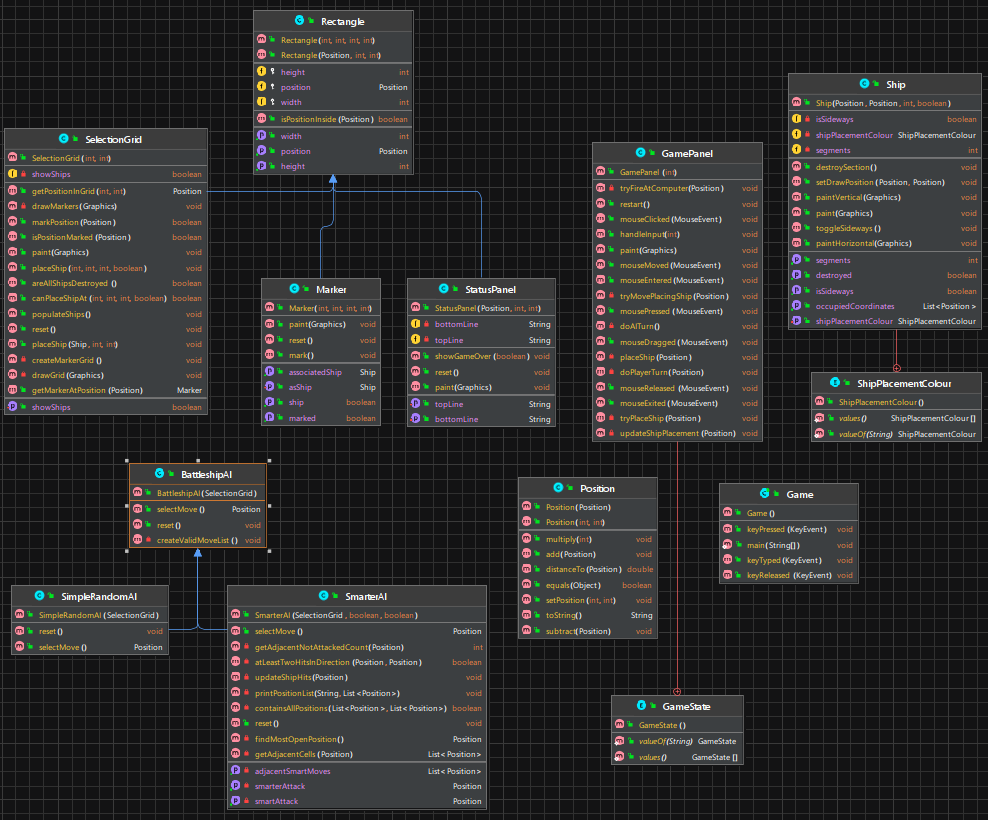
·  **StatusPanel:** Represents the text panel in the middle between the two selection grids.

**AI:**

· **BattleshipAI:** Defines the template with methods for the other classes to override. This does not do anything useful by itself.

· **SimpleRandomAI:** Defines the Easy difficulty AI that takes the list of valid moves, randomizes the order of the moves, and then selects the first option.

·  **SmarterAI:** Defines the Medium and Hard difficulty AIs. Medium AI is activated by leaving the two parameters false. Hard AI is activated by setting the two parameters to true. You could have difficulties between these two by activating just one of the two parameters. The main thing that the smarter AI does is it will focus on the area around where it finds a ship hit. The medium AI will just choose random adjacent positions, and otherwise randomly select any valid move. The hard AI with preferMovesFormingLine active will try to form lines with moves once it finds a ship hit, this improves the chance of adjacent selections also being a hit. The hard AI with maximiseAdjacentRandomisation active will change behavior while selecting moves when there is no recent ship hit. It will search for the valid move with the highest number of unmarked adjacent cells.



**4. Extra feature in future**

Here is a list of potential modifications to the existing game:

* **Enhancing the Hard AI:** The Hard AI currently has some shortcomings. For instance, if ships are placed along the edges, the AI will likely only find them after checking almost half the cells in the central region. As seen in the provided example, the AI only discovered the hit at (3, 0) when all central areas had three or fewer adjacent unmarked cells. To make the AI more challenging, we could allow it to detect edge-placed ships and subsequently prioritize edge positions. A similar issue arises when the last ship is only two segments long. Without significant "cheating" on the AI's part, solving this problem could be challenging.
* **Develop a Unique AI:** The provided AI examples can serve as a base for your own implementation.
* **Upgrade the User Interface:** The current interface is quite simple. We could add images for different types of ships and more engaging UI elements for a better experience.
* **Experiment with Grid and Ship Variations:** Test the game dynamics by altering the grid's size or the number and types of ships. This can be done by adjusting the GRID\_WIDTH, GRID\_HEIGHT, or BOAT\_SIZES variables in SelectionGrid.
* **Incorporate Features from Other Battleship Implementations:** Explore other versions of Battleship to gain inspiration and add new features to your game.