Project Report – Battleship

Student: STUPARU Andrei-Cristi

1) Instruction for compilation:

I have structured my project in 2 parts:

- BattleshipPartAI: which consists of the code necessary to run 300 games between 3 levels of intelligence of AI.
    - To compile we need to use in terminal the following command:
      javac fr/battleship/*.java
    - To run the program we need to use in terminal the following command
      java fr.battleship.TestIA
      Which will lead to the start of the program. You will be asked to insert the height and the length of the board. After that the program will start to run for 3 sets of 100 games. All the score data will be exported in the csv file attached.
- BattleshipPartHuman: which consists of the code necessary to run repetitive games either between a human and an AI or between 2 humans.
    - To compile we need to use in terminal the following command:
      javac stuparu/andrei_cristi/*.java
    - To run the program we need to use in terminal the following command
      java stuparu.andrei_cristi.Battleship
      Which will lead to the start of the program. You will be asked a couple of variables to decide what type of game you will play.

2) Architecture

A) I shall proceed in explaining the logic of the project in the same order in which it has been realized.

The first build Class was Ship, I begun the construction of the shipClass starting simple and it became more and more complex with each step. The core idea was that a Ship should have a name(shipClass), a size, a number of hit points(hp) which should be the same as its size, a head coordinate, an end coordinate and of course an owner.

By giving the name, the head, the end and the owner, the algorithm know to fill the other values. For example: if the name is “Destroyer”, the ship size and hp will be equaled with 2 and by using the head and end we can deduce the position of a ship (Vertical or Horizontal). I also store all the positions of a ship in an Array for future use.

Aside from the constructor, the all ships also have 3 other functions: isHit (which checks if a ship would be hit by a missile), hit (which realize the attack on the ship) and isDestroyed (which checks if the ship has any hp left; if hp is equal to 0, the ship will be considered sunk and

its display on board will be changed to S (from sunk) on the board of Attacks and with W (from wreckage) on the board of ship).

The second build Class was Board, which has a 2D matrix and 2 values, its height and its length. By giving the sizes of the board, the matrix will fill itself with "~" representing water or empty space.

Functions as clear and fillBoard, have the purpose of setting or resetting the board to emptiness while showBoard and showEmptyBoard display to the player either the true map or a fake, empty one.

Originally, there was a fleet Class, but after some thought, its values were introduced as part of the Player.

There is little to no difference between CPU and Human, as they are both Players, so their structure is rather similar. They have a name, a status, a board of their ships (myBoard) and a board of their attacks on the enemy (enemyBoard), they also have an enemy, either Human or CPU. Another part of the player is its fleet, which is stored in 2 arrays, one with all the ships and one with all the spaces used by the previously said ships.

As explained before, the clear function has the purpose of resetting the values of a Player in order to start another game.

Functions fill, setEnemy are used to define the basic values of the player.

Functions add, placeShip are used to fill the fleet and myBoard of the player.

Functions showBoards and showEmptyBoards are used for displaying the true and fake versions of the 2 boards of the player.

Functions hit, gotHit and gotShipSunk are used for changing the values on boards in order for the player to know its status.

In order to continue, I find it useful to explain the display of the boards a bit: a placed ship will be represented by the "#" symbol, a hit will be represented by "0" on myBoard and by "H" of enemyBoard, and a sunk ship will be displayed as "W" on my board and as a "S" in enemyBoard.

There are also some general functions as stringToInt, charToInt, getCharForNumber used for conversions of Strings in Integers, Characters in Integers or Integers into Characters.

As for the mains, each of them has a similar yet different structure.

Both of them have the check functions at the beginning for verifying if the values received are good or not. Another similarity between them is that both have a phase were the ships are created and placed and a phase of attack.

TestAI needs the length and height of the board at the very beginning of the game in order to proceed and compute the 3 sets of 100 games. The size of the board is directly

proportional with the time needed to finish the games. The game is set to get 1 of each type of ship but it can be easily changed in code by incrementing the values of limitShip1-5. After easy set is finished, the result data will be send to ai_proof.csv by the flush command.

Battleship is a bit more demanding in the data required for the game to start but it does mainly the same commands. The Player is required step by step instructions so that the game can proceed. He is given its options in matter of ships creation and has no restrictions in attacking the same spot twice, its up to him if he wants to waste a round.

A round of a player begins by telling him if he has been hit by his enemy last round, he them proceeds to attack, the winning condition is to destroy as many ships of the enemy as there are limited to the game (more exactly all of them). At the end of the round, the player tracks are hidden with fake, empty boards so that no Player can peek at the board of the enemy.

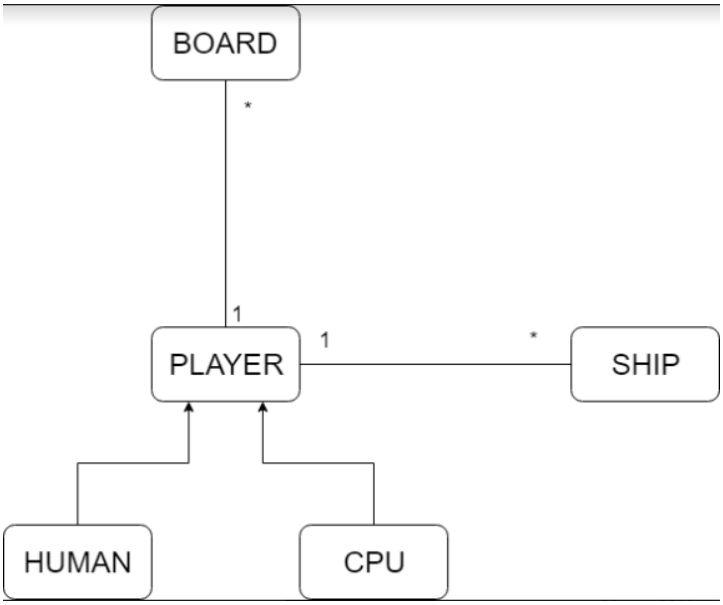
B) Pros:

- I used only public values in the project, which gives easier accessibility to the code;
- KISS (Keep It Stupid Simple) structure, I chose not to go for complicated structure and mess up or run out of time, I created only what I felt it was necessary;
- Aside from giving examples and explaining what needs to be done, I also minimalized the risk of the game crashing due to the introduction of wrong values.

Cons:

- For a real project, private values are more common, using getters and setters might have been better in the long run;
- The ship placement and attack part could have been structured in functions in order to offer reusability;
- Might have been better to create a Coordinate class rather than using Strings

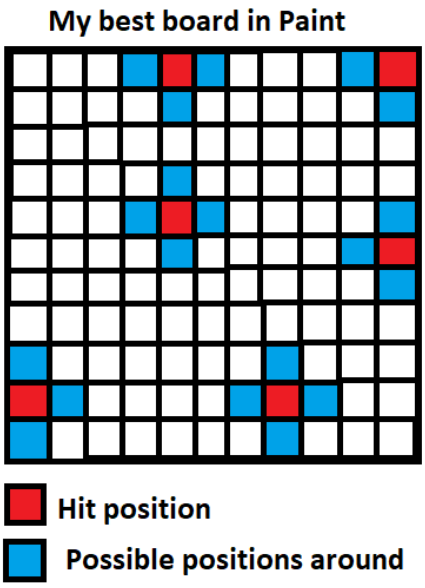
C)



3) AIs

The AIs are structured on 3 levels: Beginning, Medium and Hard, in this section I will do my best to explain the logic between each of them starting from the very basic to the most advanced.

- AI Beginner: This AI is the most basic of them all, it will randomly pick a column(A-Z) and a line and combine (1-26) these coordinates to get a point in a 2D matrix with missiles of type "A5";
  It will continue to hit randomly until all enemy ships are sunk;
  There exists a chance that he will shoot 2 or more times in the same spot.
- AI Medium: This AI is not much different from the first one as it does mostly the same thing, the only difference between these 2 is that the Medium AI stores all the hit coordinates in an Array and will make sure that it does not hit 2 times in the same spot in order to find faster the enemy ships.
- AI Hard: The biggest difference in complexity is between AI Medium and AI Hard as I built the second to approach the problem as a person would.



Firstly, he will shoot randomly in order to find an enemy ship.

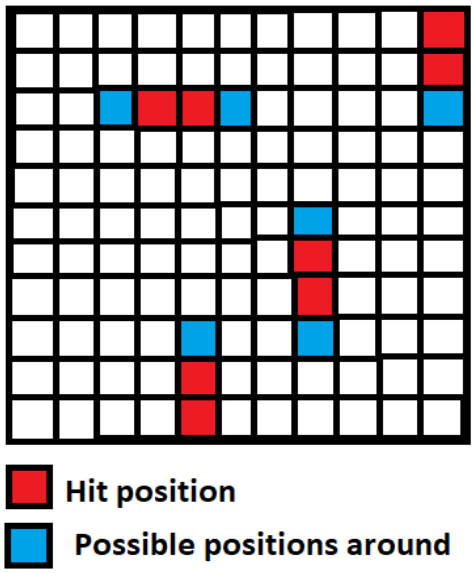
Once, he has hit a ship, he will store the value in 2 different Arrays. The value will remain permanently in the first Array (Array A) as it makes sure that the AI does not hit twice the same spot. In the 2nd Array (Array B), the value will remain only temporarily.

If there is a coordinate in B, the AI will check the position in order to shoot around it.

After the algorithm has found at least 2 spaces of the ship, considering the ship is not a destroyer and it is already sunk after the 2nd hit, it will determine the position of the ship by comparing the already hit coordinates.

If the position of the ship is Horizontal, the algorithm is supposed to continue hitting on the same line, otherwise it should hit on the same column.

Once a ship is declared sunk, Array B will clear itself so that the algorithm can search for a new boat and not check around a sunk ship.



4) Post Mortem

I feel like at least once, everything went wrong at some point. Either the display of the board or the conversions, the functions or the scanner.

On the other hand, it was good that I had ideas, it would have been worse for the code to work, but for me to have no idea of what I should do or how to do it.

Lessons I have learned:

- Do not leave unfinished work while close to the deadline (learned only the moral, not how to actually do that);
- To comment on my code better (had some problems finding the end of while loops);
- It is a good idea to not do fixes shortly before turning in the paper. (I have not actually had time to check all possible options on the code after I applied some late fixes);
- I learned how to use try-catch in order to avoid errors.