Scala Project Report

Battleship game

Laure MARCHAL
Promotion 2019
Department Computer Science & Management

Table of Contents

Instructions

To be able to play the battleship with this application, you must follow the instructions below :

After cloning the project from GitHub, you should open your terminal and go to your folder containing the repository.

```
[nena@nena-pc tmp]$ git clone https://github.com/LaureMarchal/Battleship.git
Cloning into 'Battleship'...
Username for 'https://github.com': LaureMarchal
Password for 'https://LaureMarchal@github.com':
remote: Enumerating objects: 829, done.
remote: Counting objects: 100% (829/829), done.
remote: Compressing objects: 100% (363/363), done.
remote: Total 829 (delta 297), reused 782 (delta 251), pack-reused 0
Receiving objects: 100% (829/829), 266.32 KiB | 198.00 KiB/s, done.
Resolving deltas: 100% (297/297), done.
[nena@nena-pc tmp]$ cd Battleship/
[nena@nena-pc Battleship]$ ls
build.sbt  project  README.md  src
[nena@nena-pc Battleship]$
```

You should see the 2 folders src and project, a README file and a build.sbt file.

Now, execute the sbt command :

```
[nena@nena-pc Battleship]$ ls
build.sbt  project  README.md  src
[nena@nena-pc Battleship]$ sbt
[info] Loading project definition from /home/nena/tmp/Battleship/project
[info] Updating ProjectRef(uri("file:/home/nena/tmp/Battleship/project/"), "battleship-build")...
[info] Done updating.
[info] Loading settings for project battleship from build.sbt ...
[info] Set current project to Battleship (in build file:/home/nena/tmp/Battleship/)
[info] sbt server started at local:///home/nena/.sbt/1.0/server/2467bd976c2f60104584/sock
sbt:Battleship>
```

Now, you just have to execute the run command and the application is launched :

```
Multiple main classes detected, select one to run:

 [1] battleship.AITest
 [2] battleship.Main
[info] Packaging /home/nena/tmp/Battleship/target/scala-2.12/battleship_2.12-0.1.0-SNAPSHOT.jar ...

Enter number: [info] Done packaging.
2

[info] Running battleship.Main
Hello ! You are going to play a battleship !
Press the number that corresponds to the desired play mode :
1 - Multi-player
2 - VS Easy AI
3 - VS Medium AI
4 - VS Difficult AI
mode :
```

Architecture

Conception choices

To design this application, I first did a conception phase. During this phase, I wrote a specification to describe the desired behavior of the application and which objects/classes and functions will have to be implemented in order for the application to behave correctly.
What I extracted from this specification is retrieved in this table :

Game	GameState	Player		Grid	Ship	Position	CaseType
		Human	AI				

Object Game : I chose to have a Game Object to describe the battleship game. The specification of this game, for instance the ships configuration (5 ships of different size), 2 players or the 4 different mode of the game. The idea is this object has the main functions useful to play a game, like a mode selection loop, a loop to place ships of each player and a game loop.

Case class GameState : To make things clearer, I decided to implement a case class GameState. This class is build on the 2 players and allows me to define the active player (the one who is playing) and the opponent player (the one who is waiting/is attacked). This class also implements a switchPlayers function, to exchange the active and the opponent players, to be able to play next turn easier.

Trait Player : Then, I decided to use a trait to define the behavior of a player because in this project, there are 2 different type of players. But the behavior of a player is basically always the same, he/she has to place his/her ships and then choose a target to shoot the opponent's ships. Since, there are AI and Human as possible players for this battleship, even if the behavior is almost the same, it does not mean that they are doing the exact same things.

Case class Human : To define a human player behavior, I designed a case class Human which extends from Player, to make sure the class is behaving like a player should. The biggest differences between the behavior of a human and an AI during the battleship are that the ships placing and shooting functions for a human consist a lot in getting inputs from the user, this is why I need distinct implementations of these functions.

Case class AI : To define the AIs, I decided to create 3 different case classes, one by level. Since the difference between this AIs is their shooting function, I needed 3 different implementations of the function.

Case class Grid : I designed a case class grid to describe the 2 different grids used by a player. This class represents a Matrix of dimension 10 x 10 which contains cases (Position). I use this class to store the ships and shots of a player by setting a particular case type (WATER,SHIP,HIT,MISSED). It is this class that implements the ships placement and the case update.

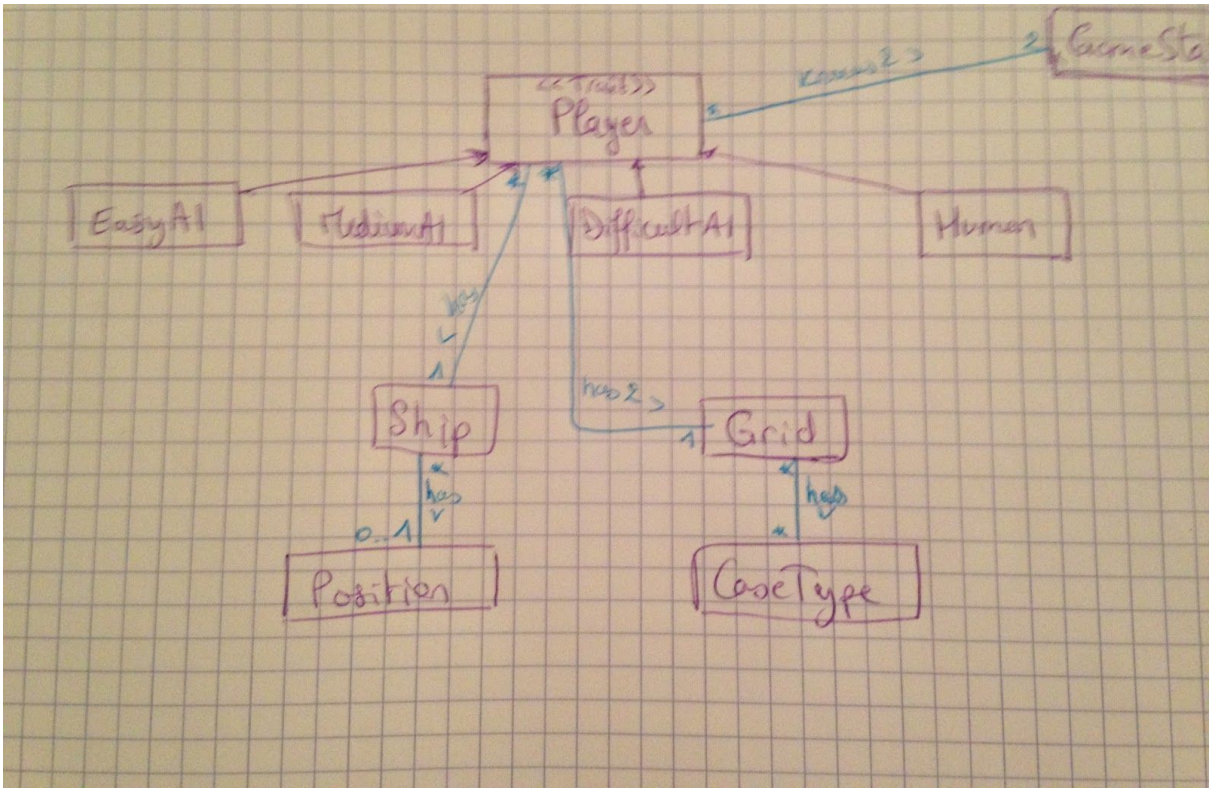
Case class Ship : This case class describes a Ship structure. A ship has some attributes like a name, a direction and a list of positions (on the grid). This class allows me to get useful information on a ship like its current size (number of position left : that have not been hit), also if a ship has been sunk or even a function to remove a position of the list of positions if the ship has been hit.

Case class Position : This case class has been implemented to define a position, which is a couple of Int. It also implements a few useful functions to get information on a position like if it is a valid position (include in the grid dimension : each coordinate is between 0 and 9) or if a position is on the limit of the grid (used by the AI to avoid shooting out when looking for the rest of a ship).

Object CaseType : I finally decided to define the possible type of case (case of the grid : a position). In the battleship game, a case can have *WATER* or a *SHIP* part for the grid containing the ships of the player. For the grid that contains the shots tried by the player, a case can be marked as *HIT* (if the initial case was a *SHIP*) or as *MISSED* (if it was *WATER*). This object is actually an Enumeration and allows me to restrict the possible values of a grid case.

Architecture Diagram

You can find the diagram on the GitHub Repository.



(I had a problem of software, sorry for this picture)

Pros and Cons

Pros	Cons
- A good maintainability because of a structure divided by functions/object	- Do not use all the possibilities of the functional programming principles - Use of mutable for some attributes

Artificial Intelligences

For this particular project, one of the requirements was to be able to play against an AI and the possibility to choose between three different levels of difficulties.
These three levels must absolutely follow one rule : the difficult level has to win against the level easy and the level medium, and the medium level has to win against the easy level.
To make sure it was the case, I decided on different algorithms for each of them in the choice of the target during a battleship game but not in the choice of the ships place on the grid. Each AI level places its ships randomly, I made this decision because of my personal experience of battleship game. I personally think that the best way to win is to choose the right target depending on the others, but that the ships grid is not that important in the victory.

AI Level Easy

To make sure that this level was easier to beat than the others, I chose to make it really simple. For this level, the AI chooses the target randomly and does not check if this target has already been tried. When it is the turn of the easy AI to shoot, the algorithm, which is behind the choice of the target, generates two random int that are between the limit of the grid. Then, it makes a Position object with the two int, checks if this is a valid position and return this position. Finally, the shoot function of the easy AI gets this position (target) and shoot without checking if this target has been tried before in the game.

AI Level Medium

Since this AI must be better than the easy and if possible, a good challenge for a human player, I decided that the choice of the target will be made more strategically. To do so, I added an attribute called lastHitShots, a List of Position, to store the last position that were hit. So, when it is the turn of this level AI to choose the target, the idea is that it will shoot a random position but only if it has not been tried yet and each time a ship is found at the target position, it will try the valid positions next to the last that was a “Hit” (the position stored at the beginning of lastHitShots) but always checking that the target has not already been tried.

AI Level Difficult

This level of AI had to be the most difficult even for a human player, it should win. The strategy I chose for this AI level is to use the same hunt than the Medium AI level but make it shoot better by using the parity for example. Because of the basic configuration of a battleship, the minimum size of a ship is 2, so without any known ship position, this AI shoots only positions with an even Int in it. This reduces a lot the grid position to try.

Post Mortem

This project was really interesting because I wanted to learn more about functional programming and this project allowed me to practise the principles of this kind of programming. It was a good experience because I learned a lot about scala language and functional concepts. Moreover, the fact that Unit Test was in the requirement of the project made me do some research about testing, which I wanted to do even before this assignment. Finally, the challenge to realize a battleship application in a short deadline was a challenge I enjoyed and would like to do again.

Like any project, there were also issues and hard times to overcome. The short deadline was not the best condition to do a good project because the programming language and the principles of this language were totally new to me. It was not easy to design the application at first because I am used to think in an object oriented way to design this kind of application. I took too much time to finish the conception phase without realising that I would need some consequent time to program the application using this new language.

At the end, I learned some lessons like even for a project of this size, a good organisation before starting programming is always useful. I also think that I could have asked more questions to the other students to get help instead of searching alone the particularities of the language and this way avoid time loss.