Python project report
Group 15: Battleship Online

CONSOLI Lorenzo GIACOPELLI Nicolo'
MAISURADZE Alexander

January 8, 2022

1 Introduction

Who does not know the Battleship Game? This game can be regarded as one of the first memorization and prediction games which is considered suitable for children of any age and any background (since the violence beneath is properly conceived). Notwithstanding its perpetual notoriety, the Game has recently bloomed once again with the rise of studies regarding Reinforcement Learning algorithms. In fact, it takes place a very structured framework but at the same time it provides very flexible game dynamics, and these characteristics make Battleship Game a perfect training gym for any new algorithm trying to find a sense of this world.

Several versions of the game exist, and in the most difficult one there is imperfect signaling about the results of the shots made by each player, so that the opponents enjoy another layer of the game (the one concerning *bluffing*). We think this could in fact be a potential development of the game that we would like to undertake, even though in the versions made so far there is perfect information regarding the results of each shot, for both players.
In general, the first two versions involve the user playing against a randomized player (that follows very intuitive game dynamics which have been developed as deep as possible in the second version) while in the third version we develop a network structure that connects two different users. We now proceed to illustrate the developments of the different versions.

2 Version 0

We conceived the first version of the Battleship Game as an illustration of the basic dynamics of the game and of the process of storage of information for the development of the game that we followed in the successive stages.
Version 0 is deprived of any GUI interface, and it involved the user directly typing the input coordinates in the terminal. The visualization is performed with the method *visualize grid* which outputs the table together with row and column labels at every stage in which the user needs some information or something has changed in the internal dynamics of the game. This information concerns either the outcomes of the opponent's shot on the Player's board or vice versa, so that an additional parameter in the function *visualize grid* allows to set a *camouflage mode* that hides the positioning of the opponent's ships.

```
!!!    THE ENEMIES ARE SHOOTING    !!!

FIUU, the enemies have missed in A, 4!
    A B C D E F G H I J
1   . . . O . . . . O O
2   . . . . . . . . . .
3   . O O O . . . . . .
4   # . . . . . O O O .
5   . O . . . . . . . .
6   . . . . . O . . . O
7   . . O . . . . . . O
8   . . O . . O O . . O
9   . . O . . . . . . O
10  . . O . . . O O . O
```

The main classes of this version are Player, Random and Ship, of which the first one is the class for the user and the second is our randomized player. The randomized player follows very intuitively the idea of a randomized search that becomes a local search whenever a ship is hit. If a shot is signaled as being a winning one, a variable *local search* is initialized to 4, i.e. the 4 possible directions along which a ship could materialize near a point (since ships cannot be placed diagonally). If the variable *local search* is 0 or the last show was not a winning one, the randomized player shots randomly among the available spots (i.e. those that have not been hit until that moment). The game itself consists in a single while loop, that runs as long as the number of ships sunk of each player is strictly lower than the total number of ships available. Following the traditional version of the game, the ships are categorized according to their different sizes (i..e their total number of lives).

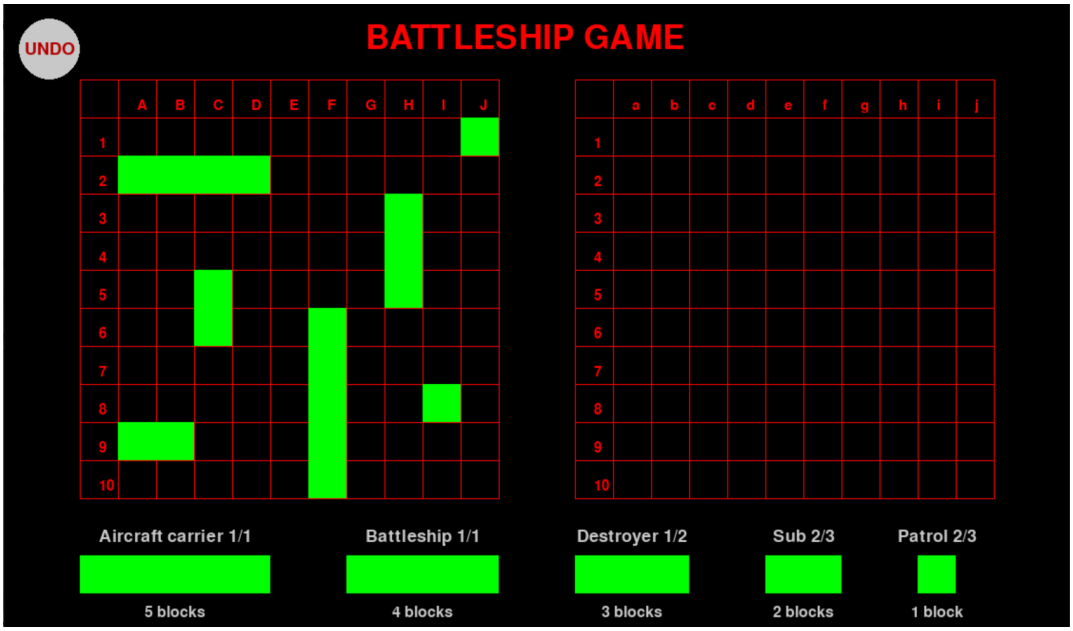
The ships available to each player are:

1. AircraftCarrier, with size 5, of which 1 instance is available
2. Battleship, with size 4, of which 1 instance is available
3. Destroyers, with size 3, of which 2 instances are available
4. Submarines, with size 5, of which 3 instances are available
5. PatrolBoats, with size 1, of which 3 instances are available

so that the total number of ships available is 10 and as long as the number of sunk ships is < 10 for each player the game continues. Importantly, this information is stored in *placed ships* which is an attribute of the class Player, together with the name of the ship concerned and the coordinates that it occupies. Starting from version 0 onward, the shadow placement has been implemented. This means that there is a strict enforcement for both players to place their ships with at least one square of distance, so that the ships cannot be placed on adjacent grid indexes. This is in fact an important information that we cared to develop since it allows a deeper rationalization of the game dynamics, and it is in fact this characteristic that differentiates the game from a completely randomized one.

3 Version 1

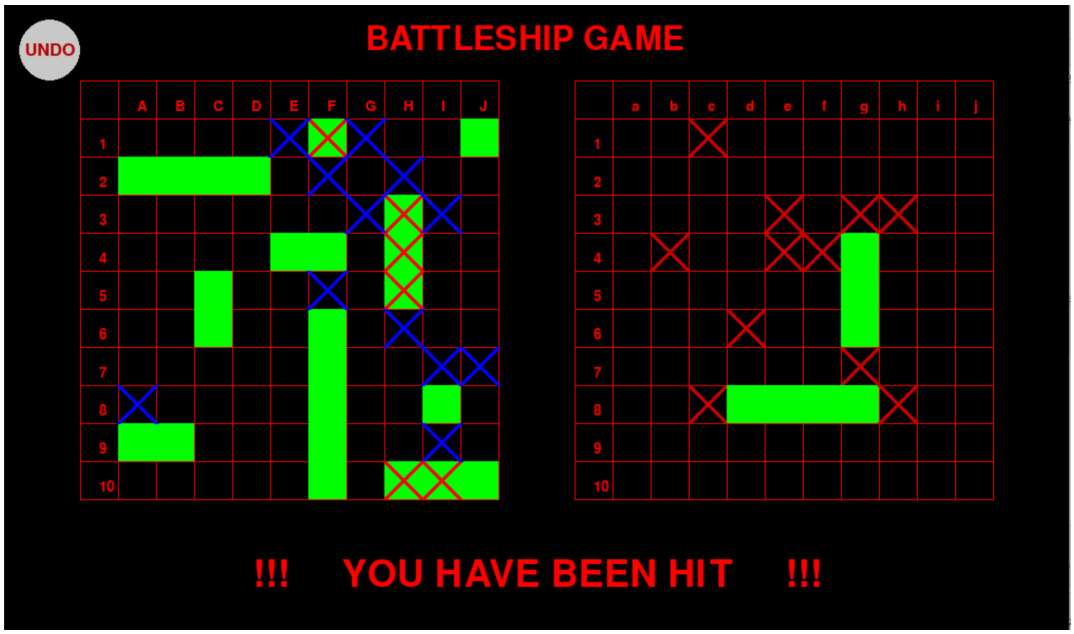
In the second version, the GUI interface is developed by exploiting the library of pygame. The instantiation of Player is now endowed with an *own board* which is an instance of the class Board. Board is an additional class that links the dynamics of the game (insertion of the ships and shots) with the graphical display of pygame.



We thought of boat insertion as the user dragging the mouse over the screen, so that in the initial phase a boat is drawn onto the Player side whenever the mouse is clicked, dragged, and released. The first and last steps identify the initial and ending index of the ship on the Board. This way of conceptualizing the game offers numerous advantages, as for example the possibility of exploiting the functionalities of pygame Groups and *group collide* whenever the mouse for a shot hits one of the opponent's ships. At the same time, the developments implemented from the previous version are kept, since the game is always supported by a purely logical structure that hinges on the numpy library for storing the information of placements/shots into a 10x10 matrix, one for each player.

The welcoming menu contains three key information on how to proceed with the game, namely how to place a ship (by dragging), how to shoot on the opponent's board (which is on the right-hand side of the screen) and a memoir of the shadow feature of the game, which is implemented graphically very intuitively with the impossibility for the player to drag, release and plot a ship lying on a square adjacent to one of the already placed ships. During the game, a UNDO button is present on the top left corner of the screen, which is

necessary since there is the possibility that given a particular placement of the ships (i.e. if a Player starts to place the smaller ones before) the user could be unable to respect the shadow feature of the game and at the same time place all ships. At the same time, this button enables for a more thoughtful placement and a enhanced strategy. The ending menu, which appears whenever the number of sunk ships of one of the two players reach 10, displays the final result and two intuitive buttons, play again and quit.

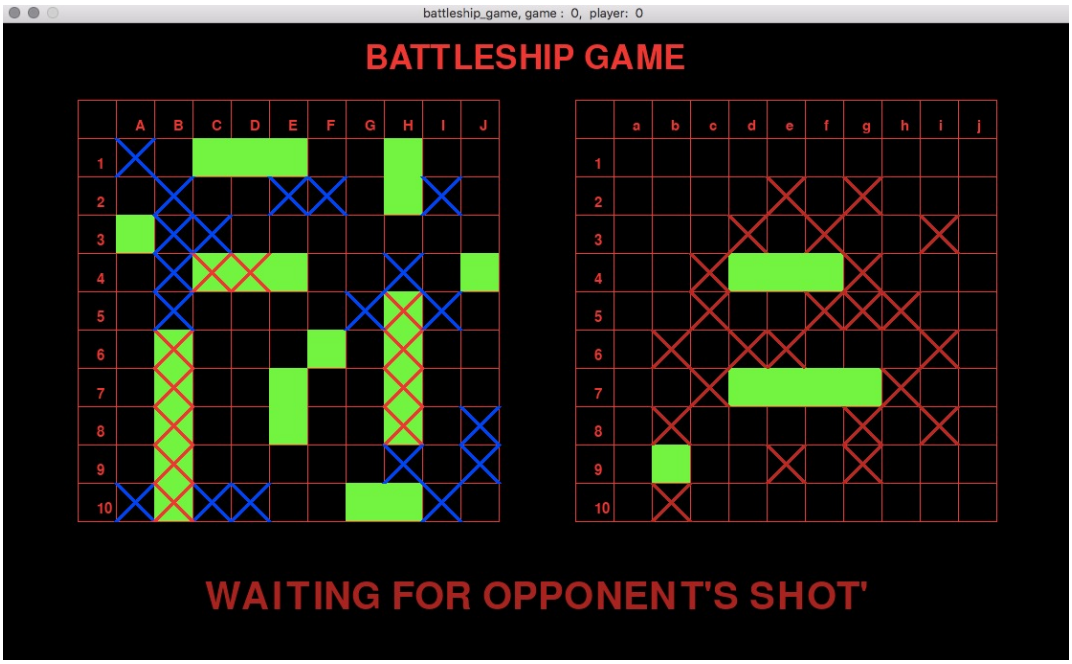


In this version, the randomized engine is developed under any possible aspect. The rationality of the game, which seems very easy for a in-person player, has been fully analyzed and implemented. The Random player is able to recognize a ship (after hitting a certain sequence of blocks and finding nothing at the extremes of this sequence) and avoid shooting around that, i.e. it is able to fully exploit the shadow feature of the game. At the same time, the localized search implemented in the previous version has been enriched so that the Random Player looks at the entirety of its historical shots in order to find close winning shots, follow a certain direction, and also follow the opposite one in the successive round in order to discard the possibility that the targeted ship actually develops in the other direction. This allows the Random player to flexibly switch between a randomized search (which neglects the squares on the shadow of the already sunk ships), a localized one (whenever a isolated square turns out to be a winning shot) and a strategic one (implementing a strategy for the successive rounds). This representation, partially graphical and partially logical, created a dissociation between the Player, endowed with a Board (which is the one and only pygame screen), and

the Randomized player which acts and is defined on a different structure. Due to this, we faced some difficulties in conciliating these two aspects of the game and making them speak to each other, so that we had the necessity to develop the Player on two different levels. This problem was actually solved with the following version, in which two different users share the same nature and play with each other across a network.

4 Version 2

In the last version, we develop a network structure with the help of the library *socket*. This allowed us to deepen our knowledge regarding Networking and all its layers, starting from the MAC addresses and the Ethernet up until the UDP and TCP protocols at the Transport layer, leaving aside all the knowledge that this implementation required regarding Routers and Servers. The idea of the socket library is the one of allowing communication across the network by sending byte-encoded strings across the Internet. This is allowed by a binding constraint between the IP address and one of the Port of a particular device, acting as a Server in fact. The sharp different between the class Player and Random in version 1 was then to be filled since, once arrived to version 2, we needed to go back and re-define some aspects of version 1.



We developed the game separately, each of us using its device as hosting both the Server and

two Clients, playing against each other. When we tried to played against each other on different devices (and different LANs) we faced some problems due to the necessity of integrating Public IP addresses instead of Private (and mobile ones). This led us to the DigitalOcean platform, where we rented a Server for a month (waiting to play with the Professor). Once we managed to connect with the Server, its Static IP address proved to be resourceful in managing the problems we faced earlier. On the Server we uploaded our server.py file (using Ubuntu) and, after having it running, we were able to face each other across the network, from one to the other part of Europe, which proved to be incredibly satisfying. Time constraints allowing, we would have liked to develop more deeply the graphical aspect of the game, even though we initially conceived the game as being a minimalist one.