Project Report - Battleship

AASMA

Bernardo Faria
MEIC-A
87636
bernardo.faria@tecnico.ulisboa.pt

Ricardo Ferreira
MEIC-A
87701
ricardo.m.s.ferreira@tecnico.ulisboa.pt

Tiago Neves
MEIC-T
90778
tiago.c.neves@tecnico.ulisboa.pt

ABSTRACT

In this paper, we will analyze the results of an agent-based project where we explored and studied multiple types of agents addressing a strategy type guessing game, called Battleship.

CCS Concepts

I.2.11 [Computing methodologies] - Artificial intelligence - Distributed artificial intelligence - Intelligent agents

Keywords

Agents; Game; Reactive; Learning; Fleet; Parity; Random

1. Project Description

Battleship is a strategy type guessing game where two players are battling against each other. At the beginning, each player has its own board and places the five ships, vertically or horizontally, in it. Players will take turns guessing by calling the coordinates they want to "shoot". The objective is to sink the opponent's ships before they do.

Our project addresses this problem, building integrated agents that play on their own, each one with different approaches. The players can be an agent or a human and the game will allow three different modes:

- Agent Only. 1-player game where the agent plays alone, shooting a board created randomly.
- Agent-Agent. 2-player game between two agents.
- Human-Agent. 2-player game between an agent and a human.

The players will progressively gain access to the opponent's world's environment as they shoot the enemy board. This information will be given to the agent through a satellite (sensor) and therefore the agent will have the information regarding if the shot hit or not.

As for the agents, there is a Naive Agent, two Reactive Agents with different approaches, and a Learning Agent.

1.1 Project Motivation

The main factor that led us to create this project comes from the desire to observe how an Agent develops strategies and how better they can be when compared to human strategies. With battleship we can easily code the strategy that a human uses to win a game and therefore compare it to the strategy that the learning agent comes up with.

Also we would be able to create different types of agents.This allows us to have multiple ways to solve the same problem and therefore evaluate the comparative results, the different approaches that the agents will address and conclude which type of agent performs better in this type of games or situations.

1.2 Project End Goal

Our project end goal is to see if the Agents can overcome the performance of a human player. We will also have an agent that uses Q-learning, in which we will try to answer if this agent has better averages than the Naive Agent and the Reactive agents. More specifically, we will use different metrics to respond to these answers.

1.3 Game Structure

The game can have two boards, or one if the game is being played in Agent only mode. Despite this, the board is one main component of the game and we decide to follow the traditional game guideline. The board is a 2d grid with size 10x10. This is where the agents will place the ships and where the players will shoot to try to hit the enemy ships.

There are 5 ships, for each player, of length 2, 3, 4 and 5, in which two of them have size 3:

- Carrier (5)
- Battleship (4)
- Cruiser (3)
- Submarine (3)
- Destroyer (2)

Also, each player has a satellite which gives information about the opponent's environment and informs the player whether a shot hit or if the ship sunk.

1.4 Game Rules

Before the game starts, each player has to place his 5 ships on the board. Players can choose to place the ships manually (the user can choose the coordinates to place the ships and choose his orientation) or randomly (the ships are placed randomly). The ships:

- Cannot be placed diagonally and therefore can only be placed vertically and horizontally.
- Ships cannot overlap and no part of the ship can be outside the grid.

After the ships are placed, the game ends when a player destroys the whole opponent's fleet. The players will alternate turns, calling out one shot per turn. Each player will decide a pair x, y which represent a location on the opponent's board and the player is informed whether the shot hits or not.

2. AI Implementation

To address this problem, four agents were built while developing the project: a Naive Agent, two Reactive Agents with different approaches, and a Learning Agent. The agents will have a satellite to inform them about the opponent's environment. Initially the environment is fully unknown and the agents will gain access to the enemy's world as it shoots.

2.1 Naive Agent

Starting with the Naive Agent, this is the simplest agent of the four. It follows a very straightforward policy: at each round, the agent chooses a random coordinate of the board. The agent accesses the satellite to see which coordinates he already shooted and chooses only coordinates where we didn't shoot.

2.2 Reactive Agents

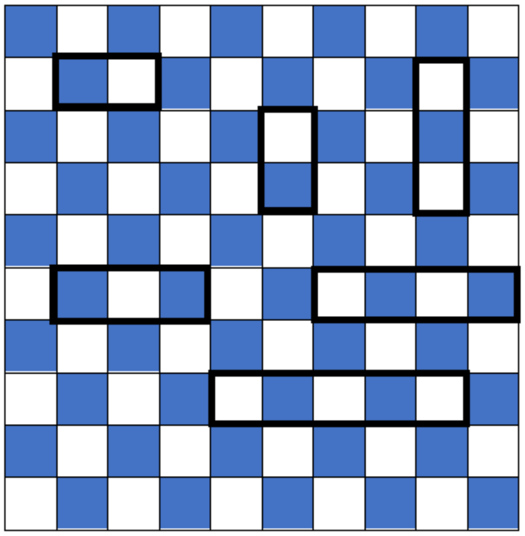
Our program has two different Reactive Agents approaches. Both policies of these two agents are more complex than the Naive Agent but they share common properties.

For both Reactive Agents, the policy works this way: the first shot of the agent is always random, by calculating an arbitrary coordinate of the board to shoot. Then, for the following shots, one of these situations happens:

- If the shot hits a ship, the agent enters exploration mode by exploring the surrounding area, namely the board cells up, down, left and right of the current location; the process repeats if one of these new coordinates hits a ship and if the cell has not yet been shooted at. All these coordinates to explore are saved by the agent.
- If the shot does not hit a ship, the agent looks for coordinates that might be able to explore. If so, pick the coordinates from the list; if not, then the agent proceeds the game by calculating a new coordinate randomly.

The policy of this agent follows the same principle that all of us usually use when playing this game. If we hit a ship, the rest of it might be in the surrounding area of the board, hence the exploration of the cell's neighborhood in order to find it. Although it also uses a random approach, this is only used when the agent has no more options to explore. However, the agent doesn't know when a ship is destroyed and therefore cannot calculate possible spaces where the ship can or cannot be. For example, if the agent knew that he already had found the ship of size 2, that is the smallest one, he could infer that two adjacent cells where the neighbor cells are not free cannot have any ship.

The only difference between these two agents is the space to shoot **when not in exploration mode**. The agent we call Reactive Random, can shoot in all of the cells. The other agent, called Reactive Parity, can only choose random cells that correspond to even cells. Even cells correspond to the ones that the sum of x and y is an even number. This slight improvement is due to the fact that the minimum length of a ship is two units long so it's not necessary to randomly search every location on the board since there is no ship of size one. Every ship, even the one with size two, has to straddle two adjacent cells as we can see in the next image.



Regarding the difference we already mentioned, that the agent doesn't know when a ship is destroyed and therefore cannot calculate possible spaces where the ship can or cannot be, we could also implement an improvement based on changing the parity. For example, once we sink the ship of size two we can change the parity restriction to a larger spacing.

2.3 Learning Agent

The learning agent works with board environment which are the states (10*10) and can choose the point to shoot where it will have to analyse a direction if it is on the side of the hit point. The agent takes the action where can study states to see which is the best direction around the hit point. For example, he knows that the closer the hit shots are, the more likely to destroy a ship and therefore get a higher reward. But also knows that around the point where the ship has already been sunk, his probability is null as by default.

In the Q Learning table, the values start always at zero and each step the agent takes an action, the table is updated with Q Learning equation that can increase or decrease depending on the shot result. The rewards have a huge impact on the table. Can be positive or negative. The agent receives a reward of 1 if he makes the shot on ship and can receive even more if his direction is correct, which is one more. If he misses the shot, receives one less. The learning agent can train several times to get the Q Learning table updated and during the battle he can make better and better decisions. It's natural for the first steps to fail a lot like the Reactive Agent with random because for first steps he usually uses random to make the decision.

3. Road Blocks

The first implementation of the Reactive Agent that follows the pattern led us to a problem, which we will explain briefly. This initial implementation followed the same cells space as the Reactive Parity agent, but with a small difference: the agent strictly followed the pattern with an approach top-down, right-to-left.. Its first shot was not a random choice from the pattern cells, but the actual first cell in the pattern (in our case, the cell (0,0)). This strict process led us to serious problems, like the agent trying to read a cell outside the game board, or even missing a large number of consecutive shots. For instance, if all five opponents' ships were placed on the bottom part of the board (to give an example, from the 7th row until the last one), the agent would fire at least 25 miss shots, since the agent strictly follows the pattern top-down. As we soon discovered this problem, we noticed that this approach was not the best one. The solution we planned for this was for the agent, in its first shot, to choose a random cell from the space environment. With this, the agent won't always start on the first cell. The example above might happen, but it is highly unlikely to, since the agent needed to choose the very first cell in the game, and at the same time, all the ships should be on the bottom part of the board. Our solution reduces this probability, enhancing the agent's efficiency.

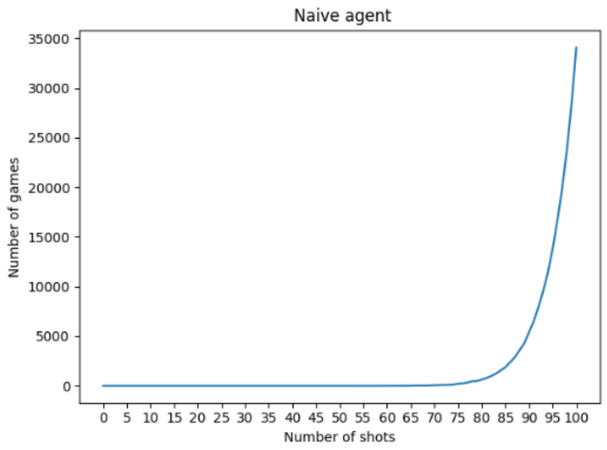
4. Experimental Analyses

In this section we will address the testing results we obtained during the analysis. At the end of the section we'll discuss the overall project result.

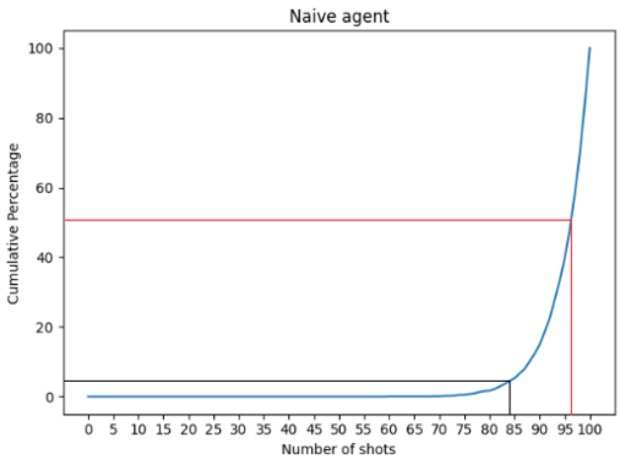
In order to evaluate the performance of our agents, we decided to guide by three metrics: number of shots per game, number of max consecutive hits per game and percentage of games won by each agent against other agents.

4.1 Metrics

Starting with the first metric, number of shots per game, we think this metric is crucial and the best to measure the overall performance of the agent. This metric will count the total shots, hit or miss, until all the fleet is sunk. We ran multiple simulations of games for each agent in order to have consistent results and then plotted the respective graphics. Every run is a new game and therefore the fleet's location is different. The x-axis will represent the number of shots and the y-axis shows the number of games that were ended x number of shots.

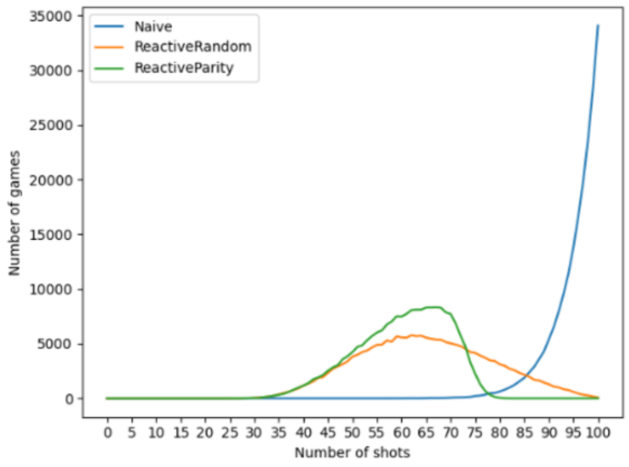


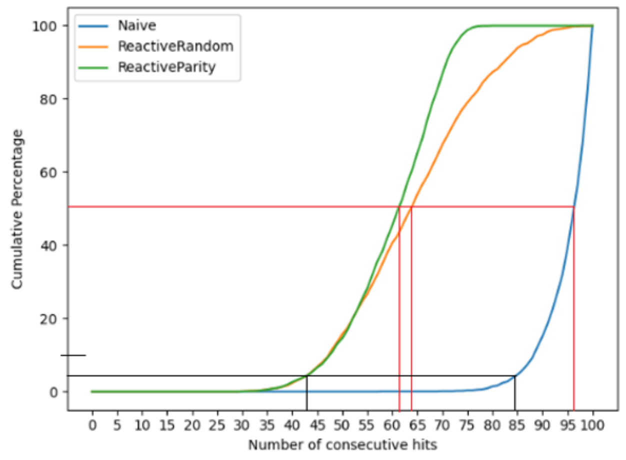
The graph above shows that the Naive agent needs more than 90 shots to end the game in the major percentage of the games.



The graph above shows the cumulative probability of completing the game with n shots. We can observe that the agent needs approximately 96 shots to complete 50% of the games and 95% of the games will take more than 84 shots.

Proceeding with the Reactive Agents, we observed a remarkable improvement when compared to the Naïve Agent. When compared to each other, they are similar but we can observe that the one with the parity random approach has a better performance than the one who considers all the cells.

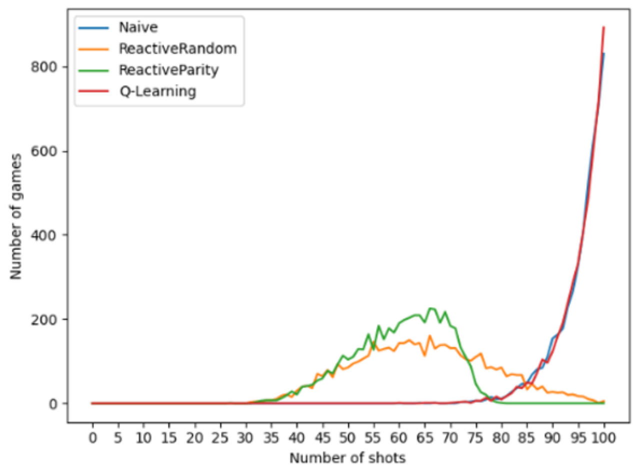




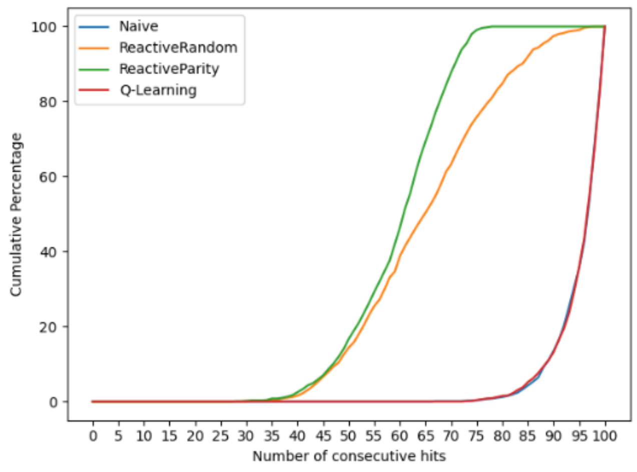
As we can see in the figure above, they both can complete games below 35 shots which is a huge improvement as we already said. However, looking at the graph, we can see that the Reactive Parity agent gives improvement over the entire range. The agent with the parity approach has more area under the line between the values 45 and 75 shots which is a consequence of the fact that he never needs more than 80 shots to complete the game. This is only due to the fact that he avoids many shots when shooting only in the even cells since they both explore the area in the same way.

The graph above shows the cumulative probability graph. We can better see the improvement of both reactive agent strategies over the pure random strategy. The Parity agent needs approximately only 61 shots to complete 50% of the games and 5% of the games will take less than 43 shots. The agent without the parity filter, needs 64 shots to complete 50% of the games as the Parity agent, also needs less than 43 shots to complete 5% of the games.

To finish the evaluation regarding this metric, we still need to analyze the Q-learning agent.

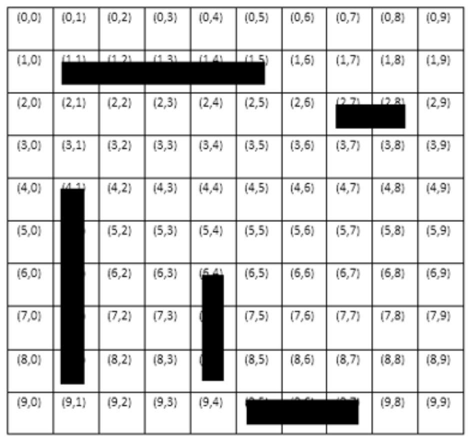


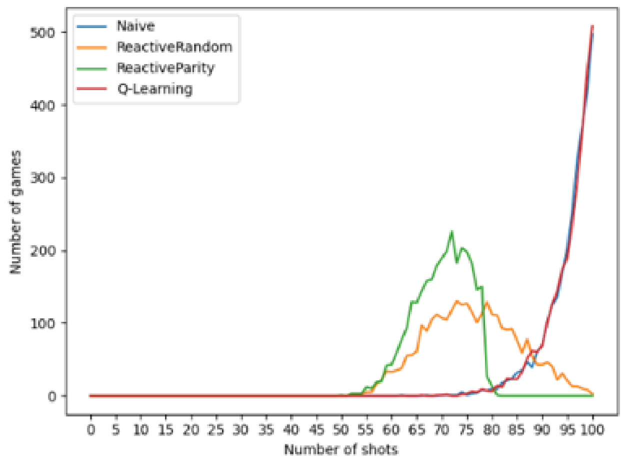
The learning agent is acting in a very similar way compared to the Naive, which makes decisions randomly. We thought that Learning could perform better as Reactive types do. We know the Learning has to use random for the first steps, but after that, it could make better decisions according to his experience.



Since in all the simulations of the previous analysis a new game was created, that means that the fleet location was always changing. Because of that, we decided to also experiment to run simulations to the where the fleet was fixed.

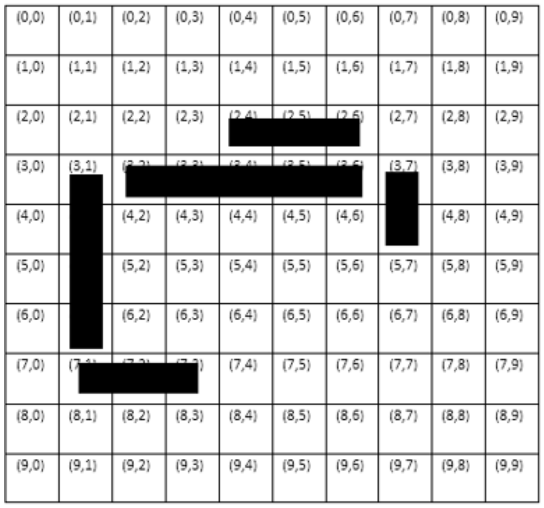
First we ran with the disposition presented on the image below. We tried to place the ships in a scattered way, to cover all the board.

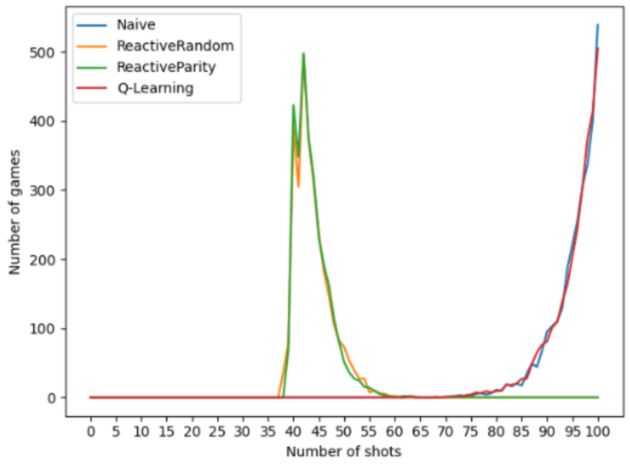




As we can see, the results were similar to the ones we already analyzed and that's good because it means that our simulations cover multiple fleet locations.

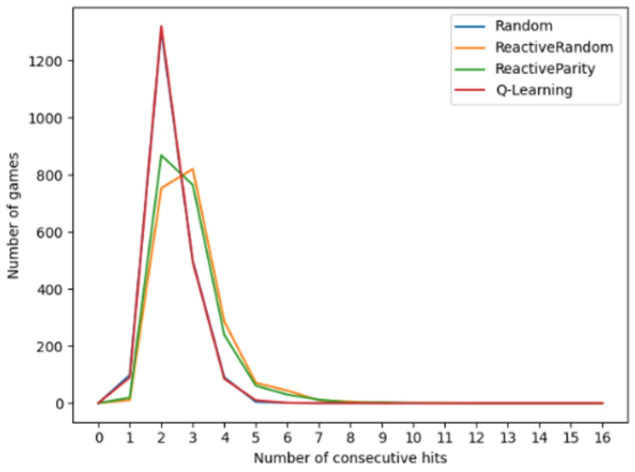
However, we decided to try a second disposition where we have the ships close to each other in a way that adjacent coordinates have different ships.





We can observe in the graphic above that the curves of Random and Learning agents didn't change but the ones from the Reactive Agents both changed. The Reactive Random agent can complete the game with 37 shots only and almost never needs more than 60. The same goes for the Reactive Parity agent. This is due the exploration mode implementation. Both these agents work better and have advantage when the ships are adjacent to each other and we can conclude by looking at the previous graphs.

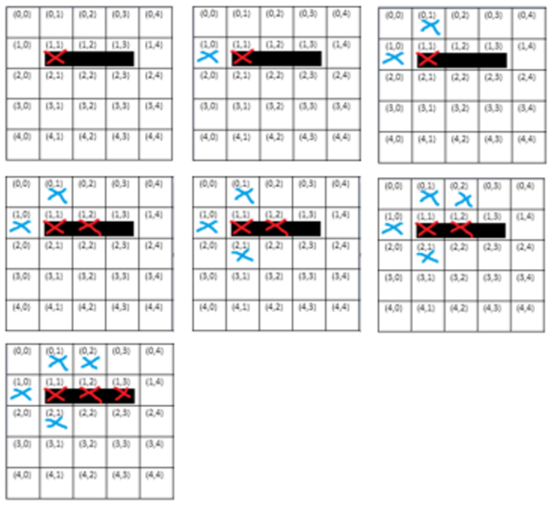
Regarding the second metric, number of max consecutive hits per game.



The above shows the number of max consecutive hits per game for each player. Once again, we have the Random agent and the Q-learning Agent with practically the same results. The majority of the games, these agents can only hit two consecutive hits and they get to hit up to 5 consecutive shots. The Reactive agents are pretty similar too, but we can observe that they hit more consecutive shots when compared with the Random and Q-learning agent. They get to hit up to 8 consecutive shots which is a very good improvement.

We can also observe that it appears that the Reactive Random agent is slightly better in this metric. The agent has a higher percentage of games in which he hits 3 consecutive shots, unlike the agent with Parity who has a higher percentage of games in which he hits 2 consecutive shots. This can be justified by the fact that an agent without parity has more cells to shoot and therefore increases the probability of hitting more shots in a row when firing randomly (when not in exploration mode).

Also, regarding both reactive agents, these values could be higher if this same idea of exploring were implemented differently. Every time a shot hits, the agent adds the neighbor cells to a list and explores. The agent explores according to the order of the coordinates in the list since he always picks the first item in the list.



In the previous image, the agent hit the cell (1,1). We will append the cells (1,0), (0,1), (1,2), (2,1) to the list. On the next iteration, he would pick (1,0) and fail. Next, (0,1) and fail. When he chooses (1,2), it's a hit and the list will have the cells (2,1), (0,2), (1,3) and (0,2). And that's where the improvement should be. Since he already explored to the left and found nothing, when he hits (1,2) he should remain exploring in that direction and only when he fails, he should change the direction. So, basically, every time he is exploring and hits, he should remain in that direction. This idea would improve the results regarding this metric.

Regarding the last metric, percentage of games won by each agent against other agents, we ran 500 simulations for each duel and we obtained the following results:

- **Random x Reactive Random**: Reactive Random won 98.6% of the games.
- **Random x Reactive Parity**: Reactive Parity won 100% of the games.
- **Random x Learning**: Learning won 60% of the games.
- **Reactive Random x Reactive Parity**: Reactive parity won 57.4% of the games.
- **Reactive Random x Learning**: Reactive Random won 100% of the games.
- **Reactive Parity vs Learning**: Reactive Parity won 100% of the games.

This metric was used only to confirm what we had already analyzed in the previous metrics. Both reactive agents are significantly better than the random agent and the learning agent. Perhaps 500 iterations are few to draw any tiebreaker conclusions, but Reactive Parity seemed to do better than Reactive Random and the Learning agent seemed to do better than Random agent.

4.2 Conclusion

Our final conclusion the following:

We envisioned a group of different AIs that were capable of resolving the game.

During our development we tried to figure out strategies that could outperform a human player. We think we achieve good results regarding the Reactive Agents but not so good regarding the Learning agent. We expected better results regarding the Q-Learning, despite the fact that not always learning algorithms can outperform random ones. Maybe we could have tried to implement a different logic, regarding the actions and the states.

We would like to try to use probability density functions in order to build an algorithm that calculates the most probable location to fire next based on all possible locations the enemy ships could be in. It would work as a heatmap. Initially all the cells would have some probability but as more shots are fired, the probability values of each cell would change, some locations become highly likely, some become less likely and others become impossible.

Overall we are fairly happy with our work, since it was our first touch with AI, and we were able to watch agents resolving the game and battling against each other.