PiratesOfTheC

November 2024

1\_ Group Members:

Jad Hdeife, Mahmoud Jebeily, Ahmad El Hariri,

Ali Wazni

2\_ GitHub URL: <https://github.com/JadHdh689/BattleShips->

3\_ High-level Description and Strategies Used:

* Difficulty chosen is based on whether or not misses appear on the grid.
* First, the bot places the ships at random validly without any issues.
* When choosing the move that the bot chooses based on the following steps:

1. The algorithm first gives priority to Torpedo.

If a torpedo can be done, the bot must choose it.

2. Then the priority goes to Artillery. If an Artillery can be done, then the bot chooses it.

3. Then, if the bot enters a hit phase (in which the bot has sunk a part of a ship) the bot chooses fire to continue the phase.

4. Then, the bot has a 50% chance of choosing fire, 25% chance of choosing smoke, and 25% chance of choosing radar. The smoke and radar have 25% chance each if the bot still can do any of them. Else, the bot chooses fire.

* For the moves Fire, Artillery, and Radar, the bot utilizes two strategies to choose which coordinates to fire at:

1. The first strategy is called Parity, and it is used at the beginning of the game until the player’s carrier (5 cells long) is sunk. Parity selects 20 specific cells such that firing at all these cells guarantees a hit on the carrier. This works because the carrier takes up 5 consecutive cells. After the carrier is sunk, the program switches to the other strategy which is called Probability.

2. In the probability strategy, the bot will fire at the cell that has the highest probability of the entire player grid. Probabilities of each cell are calculated based on how many different permutations of ships that would occupy this cell, so the more ships you can place, the higher the probability is.

* The bot calculates coordinates for the next fire differently when it reaches a hit phase.
* If the bot finds a ship that has not been sunk but has parts that have been sunk, the bot enters a hit phase on a cell that has been sunk, which helps the bot sink the ship.
* Whenever the bot hits a ship successfully for the first time, it enters a phase where it would focus on destroying the ship that it just hit.

In the phase, the bot will fire at the 4 adjacent cells of the last hit, hitting in order from highest probability to lowest, until it gets another successful hit. Then it would keep going in that direction as it keeps destroying parts of the ship until it can’t anymore (out of bounds) or it hits water, which is in when it goes back to the original hit cell (the first hit) and begins the cycle again by choosing another adjacent cell. This phase stops when the ship has been sunk. Note that when a cell is fired on, it will always have a probability of 0, meaning we cannot revisit an adjacent cell that has been fired on.

* For both these strategies, if there had been a recent Radar used successfully (ships found successfully),

the bot would fire at the 4 cells, from highest probability to lowest, until it gets a hit. After it gets a hit, it enters the regular hit phase to make sure it sinks the ship.

* For the move Smoke Screen, the bot will place the smoke screen over the first unsmoked cell it finds that has a standing ship.
* For the move Torpedo, the bot will calculate the sum of probabilities of the cells in each row and column and fire at the one with the highest probability.

4\_ Issues:

* We faced a specific case issue in which the bot might hit a part of the ship which has adjacent cells that have already been fired on, making them have a probability of 0. This leads to the bot always aiming at the same coordinates every round, since the algorithm cannot move in any direction of the original hit, and the coordinates for firing will stay stuck at the original cell that was hit. This aiming at the coordinates occurs since the algorithm does not exit the hit phase until the ship that was fired on has been sunk.
* In the parity strategy, the algorithm would choose randomly from the set of 20 cells. This means the bot might choose a cell that has already been hit, which leads to useless rounds.
* If the player places several ships that are right next to each other, this might lead to the bot to take more turns to destroy all the ships. This is since when the bot enters a hit phase, it might go in a direction where it hits other ships. After the leaving the hit phase, the ships that were hit will have a lower probability than other cells, which might in turn lead to the bot taking more turns until he fires again at the previous area.

5\_ Resolutions:

* To fix the issue of the bot getting stuck, we did a condition for the hit phase to continue. When calculating the coordinates of the adjacent cells, if at the end of calculation, the new coordinates were equal to the initial coordinates (meaning the bot got stuck), the algorithm would exit the hit phase and would calculate new coordinates based on either the parity strategy or probability strategy or the find remaining parts of a ship strategy. The bot would eventually sink the ship that has been partially broken again through these strategies.
* To make sure that the algorithm does not choose a coordinate that has already been used in the parity algorithm, we created a 1D array that included all the coordinates. The algorithm would choose a coordinate randomly by using the modulo size formula. After it chooses a coordinate, the element is swapped with the last element in the array, and the size constant is reduced by 1. In this way, we make sure that the algorithm doesn’t choose a coordinate that has already been used.
* To solve the issue of the bot taking time to sink shapes that are placed near each other, we created a separate grid called “SunkShipsGrid” that is at the start the same as the player’s secret grid(the grid with all the ships placed), and it only updates when a ship is sunk. When a ship has been sunk, the cells occupied by the sunk ship all turn to water. Then, if we have a successful hit (\*) in the player’s public grid (the grid that tracks the bots hits and misses), and on that same coordinates we have a ship in the SunkShipGrid(so we have a successful hit on a ship that has not been sunk), then the bot enters the “hit phase” around that coordinate, which allows the bot to sink the ship.

6\_ Limitations:

* The bot has completely random ship placements, which means the bot have some ships right next to each other by chance.
* Due to our implementation, the bot might lose a turn if there is multiple hits on ships that have not been sunk. The bot would do an uneducated hit based on probability before entering a hit phase around the ship.

7\_ Assumptions:

* Assume the player places all his ships correctly.

If a player misplaces a ship (out of bounds or intersection), the player will be forced to place it again.

* Assume that the bot places his shapes randomly and correctly.
* Assume the bot eventually targets all ships that the player placed, and will win if the player takes too many turns and doesn’t sink all the bots ships.
* Assume the bot does not know the placement of the ships of the player.
* Assume that the player and the bot each take turns doing their moves equally. (1 move per turn)
* Assume that the player cannot know the bot’s ship placements.
* Assume that the game automatically ends when all the ships of either the player or the bot have been sunk.