Battleships Project – Test cases

Test Case 1: Choosing the Best Move Based on Priority and Conditions

Testing Strategy: Decision-Based Testing

Description: Test the bot’s decision-making process for choosing which move to use when multiple options are available. Verify that the bot prioritizes the moves based on the following hierarchy: Torpedo > Artillery > [Radar (25%) and SmokeScreen (25%) and Fire (50%)], and ensures that the chosen move meets the required conditions for use.

Expected Outcome:

If a Torpedo is available, the bot and executes it. If a torpedo is not available the bot checks for artillery availability and executes it. If artillery is unavailable, the bot decides with a 25% probability to use radar if its conditions are met, and with 25% probability, the bot chooses to deploy SmokeScreen if it fits the conditions for defensive play. And with 50% of using Fire. If none of the above moves are chosen or conditions are met, the bot defaults to Fire.

Test Case 2: First Recent Hit

Testing Strategy: Functional Testing

Description: Test the bot's behavior when it registers its first hit on a ship. Verify that the bot correctly prioritizes attacking the cells adjacent to the hit based on probability of each cell(up, down, left, right), as this is the optimal strategy to find the full ship.

Expected Outcome: The bot saves the coordinates of the hit cell and prioritizes its neighboring cells for its next fire.

Test Case 3: Another Hit After the First Recent Hit

Testing Strategy: Scenario-Based Testing

Description: Test the bot's response when it successfully hits a ship again after the first recent hit. Check if the bot updates its strategy to focus on the new hit direction, so it should start explicitly going either left, right, up or down until it cant anymore or gets a miss.

Expected Outcome: The bot continues targeting cells along the current direction of hits, prioritizing the completion of the ship.

Test Case 4: Miss While Going in One Direction

Testing Strategy: Scenario-Based Testing

Description: Test how the bot handles a miss when pursuing a potential ship along one direction. Ensure it returns to the first hit location and starts targeting the other directions (excluding the one where it already missed).

Expected Outcome: The bot correctly identifies the missed path as invalid, resets to the original hit cell, and explores remaining directions systematically.

Test Case 5: Hitting Multiple Ships Before Sinking One

Testing Strategy: Edge Case Testing

Description: Test the bot's behavior when it hits multiple different ships placed adjacent to each other before sinking any of them. Ensure the bot can correctly sink each ship one by one while keeping track of the first hit location for each ship systematically.

Expected Outcome: After sinking any of the ships, the bot searches for remaining hits of non-sunk ships, and marks them as the original hit of a another ship, then prioritizes the cells around that hit.

Test Case 6: Using Parity to Select Cells Without a Recent Hit

Testing Strategy: Functional Testing

Description: Test the bot's behavior when there are no recent hits and it uses a parity-based strategy(before the carrier has been sunk) to decide which cell to fire at. Verify that the bot randomly fires at cells in the created parity array (contains 20 specific cells) and doesn’t repeat coordinates.

Expected Outcome: The bot randomly selects cells based in the parity array and does not target previously fired cells. This ensures efficient exploration of the grid while avoiding redundancy.

Test Case 7: Using Probability to Select Cells Without a Recent Hit

Testing Strategy: Functional Testing

Description: Test the bot's behavior when there are no recent hits and it uses a probability-based strategy(after the carrier has been sunk) to decide where to fire. Ensure the bot correctly utilizes the calculateProbability function to determine the highest-probability cell and fires at it.

Expected Outcome: The bot fires at the cell with the highest calculated probability of containing a ship. After firing, it updates the probabilities for the grid to reflect the new information and avoids targeting previously fired cells. This approach maximizes the chance of hitting a ship while maintaining efficiency.

Test Case 8: Successful Radar Hit

Testing Strategy: Functional Testing

Description: Test the bot's behavior when it registers a successful radar hit, indicating that at least one part of a ship is within the 2x2 grid covered by the radar. Verify that the bot prioritizes targeting one of the cells within the radar's coverage area, based on highest probability.

Expected Outcome: The bot selects and fires at one of the four cells within the radar's 2x2 grid. If the bot hits a ship in that cell, it transitions to targeting adjacent cells to sink the ship. If the bot misses, it continues firing at the other cells within the radar's range until it locates the ship.

Test Case 9: Continuing After Artillery Strike

Testing Strategy: Scenario-Based Testing

Description: Test the bot's behavior after using an artillery strike, which targets a 2x2 grid of cells simultaneously. Verify that if the artillery hits a ship, the bot correctly marks the last hit ship as the new "recent hit" and continues targeting cells adjacent to the hit to sink the ship.

Expected Outcome:

1. It marks the most recently hit cell as the "recent hit" and updates its targeting strategy accordingly, targeting the cells adjacent to it. Should call a “clean up” function if it runs to trouble and has all the adjacent cells around already hit. The function looks for a hit cell of a ship that isn’t sunk which has water in at least one of the adjacent cells (function is called Destroy\_remaining\_ships)
2. If no hits occur, the bot moves to its default targeting strategy (e.g., parity or probability-based selection).

Test Case 10 :Torpedo Strike Targeting a Row or Column

Testing Strategy: Functional Testing

Description: Test the bot's behavior when using a torpedo, which targets an entire row or column. Verify that the bot correctly selects the row or column with the highest sum of probabilities (calculated using the calculateProbability function) and executes the strike.

Expected Outcome:

The bot calculates the sum of probabilities for all rows and columns. It selects the row or column with the highest sum. The torpedo strikes all cells in the chosen row or column. After the strike, if a hit occurs, the bot marks the hit as the "recent hit" and adjusts its strategy to sink the ship. If no hits occur, the bot resumes its default targeting strategy (e.g., probability-based or parity-based selection).

Test Case 11: SmokeScreen Usage

Testing Strategy: Functional Testing

Description: Test the bot's behavior when using the SmokeScreen ability. Verify that the bot correctly identifies the first ship that has not yet been protected by a SmokeScreen and places it there.

Expected Outcome: The bot iterates through its own grid and checks the first ship it finds. If it already has a smokescreen placed on it, it continues iteration until it find a ship that has not been smoke screened and places the smoke screen on it.