Phoenix

November 2024

1 Group Members

Charbel Dawlabani, Elie Jbara, Crissie Tawk, Marianne Elias

2 GitHub URL

https://github.com/Dawlabani/CMPS270Project1

3 High-level Description and Strategies Used

In the realm of classic strategy games, **Battleship** has long been a staple, captivating players with its blend of tactical planning and chance. Building upon this enduring legacy, the developed Advanced Battleship Game offers an enhanced and engaging experience by incorporating additional strategic elements and sophisticated AI behaviors. Implemented in the C programming language, this game not only retains the fundamental mechanics of traditional Battleship but also introduces novel features such as Radar Sweeps, Artillery Strikes, Torpedo Launches, and Smoke Screens. These enhancements elevate the gameplay, providing players with a deeper level of strategy and interaction.

Core Features and Mechanics

At its core, the Advanced Battleship Game maintains the essential objective: players strategically place a fleet of ships on a concealed grid and take turns attempting to locate and sink their

opponent's vessels. However, the introduction of specialized actions significantly enriches the tactical possibilities.

- **Radar Sweep:** Allows players to scan a specific area of the grid, revealing the presence of enemy ships within a designated zone. This feature aids in narrowing down potential ship locations, adding a layer of reconnaissance to the gameplay.
- **Artillery Strikes:** Empowers players to execute powerful attacks targeting a 2x2 grid area. This not only increases the likelihood of hitting enemy ships but also introduces a risk-reward dynamic, as players must judiciously choose when to deploy such potent maneuvers.
- **Torpedo Launches:** Provides players with the capability to bombard entire rows or columns, offering a comprehensive offensive strategy that can devastate an opponent's fleet when used effectively.
- **Smoke Screens:** Introduces defensive tactics by allowing players to obscure sections of their grid, protecting their ships from being detected or targeted by enemy actions. This feature adds an element of deception and requires opponents to adapt their strategies accordingly.

Difficulty Levels

To ensure a challenging and dynamic experience, the game incorporates a sophisticated AI opponent with adjustable difficulty levels—Easy, Medium, and Hard. These tiers dictate the AI's strategic depth, decision-making processes, and utilization of special actions, catering to a wide range of player skill levels. The AI employs probability-based targeting algorithms, enhancing its ability to predict ship locations and respond to player maneuvers with increasing sophistication as difficulty escalates.

Strategies used

Hunt Mode

The Checkerboard Strategy is a systematic approach employed to maximize the probability of locating enemy ships with the minimal number of attacks. By targeting cells in a patterned manner akin to a checkerboard, players can efficiently scan the grid, ensuring comprehensive coverage while reducing redundancy.

Rationale Behind Effectiveness

1. Optimal Coverage with Minimal Redundancy:
    - Maximizing Detection Probability: By targeting cells in a checkerboard pattern, the AI ensures that ships of varying lengths are inevitably intersected by at least one targeted cell. For instance, a ship occupying 2 consecutive cells will always overlap with one of the checkerboard's targeted cells, so targeting only one coordinate is the most sensible option. This prevents redundant attacks.
2. Mathematical Foundation:
    - Parity Consideration: Ships occupy contiguous cells, and by targeting every other cell, the AI effectively divides the grid into two parities (even and odd). This division ensures that ships spanning multiple cells cannot evade detection by remaining entirely within one parity.
    - Probability Distribution: The Checkerboard Strategy aligns with probability theory, where covering half the grid in a structured manner provides a balanced distribution of attack probabilities across the board.
3. Efficiency in Hunt Phase:

- Transition to Target Mode: Once a ship segment is hit within the checkerboard pattern, the AI can seamlessly transition to Target Mode, focusing subsequent attacks on adjacent cells to sink the detected ship. This transition minimizes the search space and accelerates the elimination process.

This strategy is used for medium and hard mode only. I agreed to leave hunting in Easy mode randomized with the permission of the TA.

Visual Representation:

- **Checkerboard Pattern Grid:**

	A	B	C	D	E	F	G	H	I	J
1	✓		✓		✓		✓		✓	
2		✓		✓		✓		✓		✓
3	✓		✓		✓		✓		✓	
4		✓		✓		✓		✓		✓
5	✓		✓		✓		✓		✓	
6		✓		✓		✓		✓		✓
7	✓		✓		✓		✓		✓	
8		✓		✓		✓		✓		✓
9	✓		✓		✓		✓		✓	
10		✓		✓		✓		✓		✓

- **✓ Represents Cells Targeted in Checkerboard Mode**

Figure 1. Table representing the checkerboard system. This method is efficient since it prevents attacks that we know are redundant based on the fact that ships occupy contiguous cells either horizontally or vertically.

Targeting Mode

Targeting Mode is an AI operational state activated upon registering a successful hit on an enemy ship. In this mode, the AI concentrates its subsequent attacks on adjacent cells to efficiently locate and sink the identified ship.

Variations Across Difficulty Levels

Easy Difficulty

- Adjacent Cell Selection:
    - Algorithm: Sequentially targets the four orthogonally adjacent cells (North, East, South, West) relative to the initial hit.
    - Direction Handling: Does not infer ship orientation; attacks remain random among available adjacent cells.
- Hit Processing:
    - Single Hit Focus: Continues targeting adjacent cells until a miss occurs or all adjacent cells are targeted.

Medium Difficulty

- Orientation Inference:
    - Algorithm: Upon a second consecutive hit, infers the ship's orientation (horizontal or vertical) based on the relative positions of hits.
    - Limits subsequent attacks to cells aligned with the established orientation. If subsequent hit is vertical relative to the initial, the hitting continues to be vertical. The same goes for horizontal.

Hard Difficulty

- Probability-Based Targeting:
    - Algorithm: Utilizes a probability density grid to prioritize cells with the highest likelihood of containing ship segments, based on current hit data and ship orientations. The algorithm would initially increment the hit target by
    - Resource Allocation: Integrates with other AI strategies (e.g., Radar, Artillery) to enhance targeting precision.
- Step-by-step example
    - **1. Identifying Possible Ship Placements**
        - Given the hit at **D5**, the bot evaluates all possible **Battleship** placements that include **D5**. The **Battleship** can be placed either **horizontally** or **vertically**.
        - **A. Horizontal Placements Including D5**
            - **Placement 1:** C5, D5, E5, F5
            - **Placement 2:** B5, C5, D5, E5
            - **Placement 3:** D5, E5, F5, G5
        - **B. Vertical Placements Including D5**
            - **Placement 4:** D3, D4, D5, D6
            - **Placement 5:** D4, D5, D6, D7
            - **Placement 6:** D5, D6, D7, D8
    - **2. Calculating Probability Increments (ΔP)**
        - Each valid placement that includes **D5** will contribute to the probability scores of its constituent cells. The **Probability Increment (ΔP)** varies based on whether the placement overlaps with known hits.

- **Assumptions:**
    - **Known Hits:** Only **D5** is a known hit.
    - **Known Misses:** No known misses around **D5**.
    - **Applying ΔP:**
        - **For Cells with Overlapping Hits (D5):** ΔP = 10
        - **For Other Cells in the Placement:** ΔP = 1

- **3. Updating the Probability Grid**
    - After evaluating all possible placements, the bot updates the probability grid by adding the respective ΔP values to each cell.

Cell	ΔP from Placements	Total P(x, y)
B5	1	1
C5	1 (Placement 1) + 1 (Placement 2) = 2	2
D3	1	1
D4	1 (Placement 4) + 1 (Placement 5) = 2	2
D5	10 (all placements overlapping D5) * 6	60
D6	1 (Placement 4) + 1 (Placement 5) + 1 (Placement 6) = 3	3
D7	1 (Placement 5) + 1 (Placement 6) = 2	2
D8	1	1
E5	1 (Placement 1) + 1 (Placement 3) = 2	2
F5	1 (Placement 1) + 1 (Placement 3) = 2	2
G5	1	1

Figure 2. Incremented probability distribution of targets to be hit in targeting mode (Hard Difficulty).

- **4. Applying the scores**
    - The bot selects the cell with the highest probability score to maximize the chances of hitting the ship.
        - **Highest P(x, y):** D5 (already a hit)

- **Next Highest Scores:**
    - **D6:** 3
    - **C5, D4, E5, F5, D7:** 2
    - **Others:** 1
- **Decision:**
    - Since **D5** is already hit, the bot prioritizes **D6** as the next target.

*The reason that I decided to multiply D5 by its overlapping hits * 10 is to offer an easy solution to the issue of retargeting hit coordinates. That way, the bot acknowledges that this cell is to be skipped, regardless of how many subsequent overlaps there are.*

Fire

The mechanism of fire is very well explained in the two previous subheadings. As previously mentioned, in *Easy Difficulty*, the bot randomly targets coordinates, in *Medium*, it uses both hunt and *Medium Difficulty*-targeting mode, and in *Hard*, it uses both hunt and *Hard Difficulty*-targeting mode. It is important to note that we avoid retargeting coordinates that were already fired upon by recording hit coordinates in the grid. This is for both easy and hard game modes.

Artillery

Artillery is a specialized attack feature in the Advanced Battleship Game that allows players to target a **2x2** grid area in a single move. This concentrated attack increases the probability of hitting multiple ship segments simultaneously, thereby enhancing the AI's efficiency in locating and sinking enemy ships.

Usage Frequency by Difficulty Level

The implementation of Artillery varies across the game's difficulty levels—**Easy**, **Medium**, and **Hard**—to provide scalable challenges and strategic depth.

1. **Easy Difficulty**
    - **Usage Frequency: 10%** chance per turn.
    - **Behavior:**
        - **Infrequent Deployment:** Artillery strikes are used sparingly.
        - **Random Targeting:** The AI selects random coordinates for artillery without strategic prioritization.
        - Easy targeting mode is used after a hit
2. **Medium Difficulty**
    - **Usage Frequency: 30%** chance per turn.
    - **Behavior:**
        - **Moderate Deployment:** Artillery strikes are employed more regularly.

        - **Strategic Targeting:** The AI targets areas identified through initial scanning strategies like the Checkerboard Pattern.
        - **Enhanced Integration:** Combines Artillery with other targeting modes to improve hit rates and ship sinking efficiency.
3. **Hard Difficulty**
    - **Usage Frequency: 50%** chance per turn.
    - **Behavior:**
        - **Frequent Deployment:** Artillery strikes are a key component of the AI's attack strategy.
        - **Heuristic Targeting:** Uses a method to hit areas with no existing hits in them.

Radar

Radar is a specialized reconnaissance feature in the Advanced Battleship Game that allows players to scan a specific **2x2** grid area to detect the presence of enemy ships within that zone. This strategic tool enhances the AI's ability to gather intelligence about the opponent's fleet placement, thereby informing subsequent attack decisions and improving overall targeting efficiency.

Usage Frequency by Difficulty Level

The implementation and strategic deployment of **Radar** vary across the game's difficulty levels—**Easy**, **Medium**, and **Hard**—to provide scalable challenges and optimize gameplay complexity.

1. **Easy Difficulty**
    - **Usage Frequency: 10%** chance per turn.
    - **Behavior:**
        - **Infrequent Deployment:** Radar scans are utilized sparingly, primarily as occasional reconnaissance rather than a core strategy.
        - **Random Targeting:** The AI selects random coordinates for radar sweeps without prioritizing high-probability areas.
2. **Medium Difficulty**
    - **Usage Frequency: 30%** chance per turn.
    - **Behavior:**
        - **Moderate Deployment:** Radar scans are employed more regularly to gather intelligence about enemy ship placements.
        - **Strategic Targeting:** The AI prioritizes radar sweeps based on initial scanning strategies like the checkerboard pattern.
3. **Hard Difficulty**
    - **Usage Frequency: 50%** chance per turn.
    - **Behavior:**
        - **Frequent Deployment:** Radar scans are a fundamental component of the AI's reconnaissance strategy, used extensively to map out enemy fleet positions.

- **Probability-Based Targeting:** Radar updates the probability grid by marking the coordinates with X in areas with no ships.

Smoke

Smoke is a defensive feature in the Advanced Battleship Game that allows players to obscure specific areas of their grid. When deployed, a Smoke Screen conceals a **2x2** grid area, rendering any radar sweeps within this zone ineffective for a limited duration. This strategic tool enhances defensive capabilities by protecting valuable ship placements and adding an additional layer of tactical depth to gameplay.

Usage Frequency by Difficulty Level

The implementation and strategic deployment of **Smoke Screens** vary across the game's difficulty levels—**Easy**, **Medium**, and **Hard**—to provide scalable challenges and enhance gameplay complexity.

1. **Easy Difficulty**
    - **Usage Frequency: 10%** chance per turn.
    - **Behavior:**
        - **Infrequent Deployment:** Smoke Screens are used sparingly, primarily as occasional defensive measures.
        - **Random Targeting:** The AI selects random coordinates for deploying Smoke Screens without prioritizing high-threat areas.
2. **Medium Difficulty**
    - **Usage Frequency: 30%** chance per turn.
    - **Behavior:**
        - **Moderate Deployment:** Smoke Screens are employed more regularly to protect critical ship placements.
        - **Strategic Targeting:** The AI deploys smoke in areas with as many ships in them.
3. **Hard Difficulty**
    - **Usage Frequency: 50%** chance per turn.
    - **Behavior:**
        - **Frequent Deployment:** Smoke Screens are a fundamental component of the AI's defensive strategy.
        - **Same technique as medium.**

Torpedo

Torpedoes are advanced offensive tools in the Advanced Battleship Game, enabling players to execute powerful attacks that target entire rows or columns on the opponent's grid. Unlike standard single-cell attacks, torpedoes allow for comprehensive assaults, increasing the likelihood of hitting multiple ship segments simultaneously. This strategic feature enhances both offensive capabilities and gameplay complexity.

Usage Frequency by Difficulty Level

The implementation and strategic deployment of **Torpedoes** vary across the game's difficulty levels—**Easy**, **Medium**, and **Hard**—to provide scalable challenges and enhance gameplay dynamics.

1. **Easy Difficulty**
    - **Usage Frequency: 10%** chance per turn.
    - **Behavior:**
        - **Infrequent Deployment:** Torpedoes are used sparingly, primarily as occasional offensive maneuvers rather than core strategies.
        - **Random Targeting:** The AI selects random rows or columns for torpedo strikes without prioritizing high-probability areas.
        - **Limited Strategic Integration:** Torpedoes serve as supplementary attacks with minimal impact on overall offensive strategy.
2. **Medium Difficulty**
    - **Usage Frequency: 30%** chance per turn.
    - **Behavior:**
        - **Moderate Deployment:** Torpedoes are employed more regularly to disrupt enemy ship placements.
        - **Strategic Targeting:** The AI prioritizes rows or columns identified through initial scanning strategies like the Checkerboard Pattern or areas with previous hits.
3. **Hard Difficulty**
    - **Usage Frequency: 50%** chance per turn.
    - **Behavior:**
        - **Frequent Deployment:** Torpedoes are a fundamental component of the AI's offensive strategy.
        - **Heuristic Targeting:** Utilizes a method to target areas with least hits in them

4. Issues

During the development of the Battleship game, several significant challenges emerged that affected both the gameplay mechanics and the overall functionality of the program. These issues primarily centered around handling edge cases for special moves that operate over a 2x2 grid

area, designing the bot's intelligent behavior to make it a formidable opponent, managing the bot's deployment of smoke screens effectively, and addressing the unintended side effects that occurred when modifying interdependent game functions.

Handling Edge Cases in Special Moves

A major issue was the proper handling of edge cases for special moves like smoke screens, radar sweeps, and artillery strikes. These moves affect a 2x2 area on the game grid, which posed problems when the target area was located near the edges or corners of the grid. In such cases, part of the intended area would fall outside the grid boundaries, leading to potential out-of-bounds errors or incomplete execution of the move.

For instance, if a player attempted to deploy an artillery strike at coordinate J10 (the bottom-right corner of the grid), the program needed to account for the fact that a 2x2 area centered on this coordinate would extend beyond the grid's limits. Failing to handle these edge cases could result in the game crashing or the move not functioning as intended, thus disrupting the gameplay experience.

Bot's Deployment of Smoke Screens

Effectively managing the bot's use of smoke screens presented another layer of difficulty. The challenge lay in programming the bot to deploy smoke screens over areas where its ships were located to reduce the player's ability to detect them using radar sweeps. However, keeping track of the bot's ship placements and deciding when and where to deploy smoke screens without making the bot's behavior predictable was complex.

Deploying smoke screens indiscriminately could lead to them being placed over empty areas, rendering them useless and potentially giving the player unintended hints about where ships were not located. Conversely, always deploying smoke screens immediately after a ship was hit could signal to the player that a ship was in that vicinity.

The difficulty was in programming the bot to make strategic decisions about smoke screen deployment that protected its ships effectively while maintaining a level of unpredictability to keep the game challenging for the player.

Interdependency and Conflicts Between Game Functions

An ongoing issue was the interdependency of game functions, where modifying one method could lead to unintended consequences in others. This was particularly evident with the smoke screen and radar sweep functionalities. Adjustments made to the smoke screen mechanics, such as changing how the 2x2 area was calculated or stored, sometimes caused the radar sweep function to malfunction.

For example, after modifying the smoke screen deployment to correctly handle edge cases, the radar sweep might begin to incorrectly identify areas covered by smoke screens or fail to detect ships in areas that were not actually obscured. This conflict arose because both functions relied on shared data structures and needed to interpret the game grid consistently.

These unintended side effects made debugging challenging, as fixing one issue could inadvertently introduce new bugs elsewhere. Ensuring that all game functions interacted seamlessly and maintained the integrity of shared resources required meticulous attention to detail and thorough testing.

Example of Conflicting Functionalities

A specific example of this issue occurred when we attempted to fix a bug in the smoke screen deployment logic. The smoke screen was supposed to prevent the radar sweep from detecting ships within its area of effect. However, after modifying the smoke screen function to handle edge cases properly, we found that the radar sweep no longer functioned correctly.

The radar sweep began to either ignore smoke screens entirely or incorrectly assume that larger portions of the grid were covered by smoke, even when they were not. This led to situations where the player could not detect ships in areas that should have been clear or received misleading information about the presence of enemy ships.

5 Resolutions

To address these challenges, we implemented several solutions that involved refining our code logic, enhancing our algorithms, and improving our testing procedures.

Handling Edge Cases in Special Moves

To resolve the issues with edge cases in special moves, we introduced boundary checks and adjusted the calculations for the areas of effect to ensure they remained within the grid's limits. Specifically, we created a function called handleEdgeCoordinates that adjusted the starting and ending indices for the x and y coordinates based on the grid boundaries. For example, when a special move like an artillery strike was initiated at the edge of the grid, the function ensured that the area of effect did not exceed the grid's size by setting the coordinates to the maximum allowable values. This prevented out-of-bounds errors and ensured that the moves executed correctly regardless of their position on the grid. We also standardized the way each special move calculated its area of effect, using consistent logic across the smoke screen, radar sweep, and artillery strike functions. This consistency reduced the likelihood of errors and made the code easier to maintain.

Mapping Out the Bot's Intelligent Behavior

To enhance the bot's intelligence, we developed algorithms that allowed it to adapt its strategy based on the game state. After the bot scored a hit, it would add the adjacent coordinates to a list of potential targets. This approach enabled the bot to focus on sinking ships efficiently by targeting surrounding areas where the remainder of the ship might be located. We also implemented a probability-based targeting system where the bot prioritized coordinates that had a higher likelihood of containing a ship based on previous hits and the remaining ships' sizes. The bot used radar sweeps strategically when it had no immediate targets, helping it to locate ships more effectively without relying solely on random guesses. To prevent the bot from

becoming too predictable, we introduced randomness into its decision-making processes, such as occasionally choosing to deploy a smoke screen or use a special move at different times rather than always following a set pattern.

Bot's Deployment of Smoke Screens

We improved the bot's smoke screen deployment by programming it to monitor the status of its ships and the player's recent actions. The bot now deploys smoke screens in areas where its ships are located, especially if the player has recently targeted nearby coordinates. This defensive strategy helps protect the bot's ships from detection without making its behavior too predictable.

To keep track of its ship placements, the bot maintains an internal map of its grid, allowing it to identify critical areas that need protection. We also randomized the timing of smoke screen deployment to prevent the player from easily deducing the bot's ship locations based on when and where smoke screens appear.

Interdependency and Conflicts Between Game Functions

To resolve the conflicts between interdependent functions, we modularized our code and clearly defined the interfaces between different functionalities. By isolating the logic for smoke screens and radar sweeps into separate, well-defined functions, we minimized the risk of unintended side effects when changes were made. We also implemented comprehensive unit tests for each function, allowing us to detect and fix bugs more efficiently. When modifying one function, we ran these tests to ensure that other functions continued to operate correctly. This approach helped

us identify issues early in the development process and maintain the integrity of the game's overall functionality.

6 Limitations

Despite the comprehensive features and strategic depth incorporated into the Battleship game, there are several limitations that affect the overall gameplay experience and player engagement. One significant limitation is the predictability of the AI's behavior. The AI tends to use all its radar functions at the very beginning of the game, deploys smoke screens as soon as possible, and utilizes artillery strikes and torpedo attacks whenever they become available. While this aggressive strategy makes the AI a competitive and formidable opponent, it also introduces a level of predictability that experienced players might exploit. ***This design choice was intentional in easier modes to keep the game exciting and to ensure that the AI poses a significant challenge by attempting to cloak its ships quickly to prevent the player from winning easily and by maximizing its offensive capabilities early on.*** However, the consistent pattern of the AI's actions can reduce the challenge over time, as players learn to anticipate and counter its strategies.

Another limitation lies in the game's pacing after all special moves have been utilized. Once the radar sweeps, smoke screens, artillery strikes, and torpedo attacks have been exhausted, the game often devolves into checkerboard firing across the grid. This shift can make the late-game experience less exciting and more reliant on probability rather than strategy.

To address this issue, the game could benefit from introducing hints or assistance mechanisms that help players locate enemy ships in the late game. My suggestion would be to offer a limited number of hints after a certain number of consecutive misses could maintain player interest and

reduce frustration. Additionally, implementing a feature where new ships arrive or sunk ships are revived after sinking a specific number of opponent ships could add a dynamic element to the gameplay. Trading current special moves like smoke screens and artillery strikes for this new mechanic might balance the game and refresh the strategic options available to players.

While the Battleship game offers a range of strategic features and an AI designed to be competitive, the predictability of the AI's behavior in easy mode and the lack of late-game excitement are notable limitations. Enhancing the AI to exhibit more varied and adaptive strategies would increase the challenge and replayability. Additionally, introducing new mechanics in the late game, such as hints or ship revivals, could maintain player engagement and add depth to the gameplay. By refining these aspects, the game can provide a more dynamic and engaging experience that appeals to both new and experienced players, encouraging repeated playthroughs and long-term enjoyment.

7 Assumptions

In the development of the Battleship game, several key assumptions were made to align the code implementation with the intended game design and mechanics. These assumptions guided the handling of special moves, input validation, AI behavior, and the presentation of game information to the players, ensuring a cohesive and balanced gameplay experience.

One primary assumption is that the smoke screen special move remains active for only one radar sweep. This means that once a smoke screen is deployed, its effect is temporary, obscuring the player's radar scans for a single turn. After this turn, the smoke screen dissipates, allowing normal radar functionality to resume. This limitation was implemented to balance the defensive

advantage provided by the smoke screen, preventing it from permanently hindering the player's ability to detect ships and maintaining the game's strategic equilibrium.

Another critical assumption is that any invalid input entered by a player results in the loss of their turn. This rule was established following the instructions provided by Dr. Zalghout, ensuring that players adhere strictly to the expected input formats and commands. While this approach increases the game's difficulty by penalizing errors, it also reinforces the importance of accurate input and strategic planning. Players must carefully consider their moves to avoid forfeiting their turns, thereby enhancing the game's challenge and engagement.

Error handling was a significant focus area, particularly concerning the placement of ships on the map and targeting already targeted tiles. It was assumed that robust error checking mechanisms would be in place to prevent players from placing ships outside the grid boundaries or overlapping ships. Additionally, the game was designed to detect and respond appropriately when a player attempts to target a coordinate that has already been fired upon. This ensures the integrity of the gameplay by preventing redundant actions and maintaining a clear and fair game state.

For special moves such as radar sweeps, artillery strikes, and smoke screen deployments, it was assumed that these moves would only accept coordinates positioned at the edge of the map, such as J9. This design choice ensures that the area-of-effect mechanics of these moves are consistently applied without exceeding the grid's limits. By restricting the activation of these special moves to edge coordinates, the game maintains predictable and manageable areas of impact, simplifying both implementation and player strategy.

The AI bot was programmed with a prioritized system for utilizing special moves, following the sequence: smoke screen, torpedo, artillery, and finally, standard fire. This priority system was

based on the strategic importance of each move, with smoke screens offering defensive capabilities, torpedoes providing powerful offensive strikes, and artillery strikes serving as area-of-effect attacks. By adhering to this hierarchy, the bot was designed to maximize its effectiveness in both protecting its fleet and attacking the player's ships, thereby presenting a challenging opponent that balances offense and defense.

In terms of game presentation, it was assumed that each player would only be able to view the map of their opponent, not their own map. This design choice maintains the classic element of Battleship, where players must deduce the locations of enemy ships without revealing their own ship placements. Furthermore, when a smoke screen is deployed, the game does not disclose the exact coordinates of the deployment. Instead, the map is immediately hidden, adding an element of mystery and preventing players from gaining information about the bot's defensive maneuvers. This approach enhances the game's strategic depth, as players must infer the presence of smoke screens and adjust their strategies accordingly without direct knowledge of their locations.

These assumptions collectively shaped the game's mechanics and user experience, ensuring a balanced and engaging Battleship game. By limiting the duration of smoke screens, enforcing strict input validation, prioritizing special moves for the AI bot, and controlling the visibility of game maps, the development team was able to create a game that is both challenging and faithful to the traditional Battleship gameplay.