Universidad Nacional Autónoma de México
Facultad de Ingeniería
División de Ingeniería Eléctrica (DIE)

Artificial Intelligence (0406)

Professor: Dr. José Jaime Camacho Escoto
2026-1 Semester

1st Lab Report
Battleship

Por:
León Vargas Luis Guillermo
Medrano Solano Enrique
Ramírez Valdovinos Eric
Rodríguez Zamora Joshua

September 8th, 2025. Cd. Universitaria.

Abstract

For this project we create a program using classes to develop two different AI agents that play the game Battleship. One of them is an example of a simple reflex agent that just checks for the immediate environment that it is in, in other words, it doesn't take care about previous movements. This agent is an instance from the class SimpleAgentReflexPlayer(). His intelligence is limited to if the previous shot contacted an enemy ship, but if the following doesn´t, it continues shooting randomly.

By the other hand, we prepared another agent, a goal based one, which is an instance from the class GoalBasedAgentPlayer(), who without doubts demonstrated being more capable for taking decisions with a specific goal, and sank the enemy fleet. For this class we used the behavioral design pattern of State, in this one we have two states / modes, ‘HUNT’ and ‘TARGET’. In base of each one the player will look to hit one the enemies boats or try to finish of sink it, respectively. Plus of that, the goal based agent implements a pattern for attack at first attacks just the pair cells randomly until find a boat, this way we can cover more space in less movements.

Also, there is another class of player called HumanPlayer that allows us to test our agents in first person, which means, play with them.

Finally, the results showed us that the goal based agent was clearly superior than the simple reflex one.

Goals

Create a program that can play Battleship using a simple reflex agent and, separately, a goal-based agent. The program must be played in two different ways:

a) Against a human.
b) Face the two AIs.

Introduction

Artificial Intelligence (AI) is difficult to define because its conceptualization has changed throughout history and often depends on perspective. Generally, AI seeks to imitate or replicate human thought processes, reasoning, and behavior. Therefore, AI can be understood through four main categories. While each category offers a valid perspective on its own, together they form a more complete definition:

a. **Thinking humanly**

Is based on cognitive science, the field that studies the human mind to construct precise and testable theories about its workings. Using this knowledge, we can create algorithms that solve problems in the same way a human would.

b. Thinking rationally

The "thinking rationally" approach is based on the study of logic, often called the “laws of thought.” However, there are two main obstacles. First, it is difficult to translate informal knowledge into the formal language of logic, particularly when dealing with uncertain information. Second, there is a significant gap between solving a problem “in principle” and solving it in practice. Even problems with a relatively small number of facts can exhaust a computer's resources without guidance on which reasoning steps to prioritize.

c. Acting humanly

This approach is centered on the Turing Test, a trial where a human interrogator must determine whether responses are from a human or a machine. Initially, the test was limited to text-based interactions. The modern standard, however, requires a "Total Turing Test," which adds the ability to perceive and manipulate objects to the original requirements of understanding language, storing knowledge, reasoning, and adapting. Despite its fame, some researchers consider the test a distraction, focusing instead on the underlying principles of intelligence rather than on simply replicating a human.

d. Acting rationally

An agent that acts rationally is essentially an AI that develops the most optimal choice (at least in performance) based on the rules, the state and the information that can be understood from the environment and the expected goals. As the AI becomes more rational, it also becomes more effective in reaching the goals. These agents can learn from every experience they have, creating adaptability in dynamic environments.

AI agents

An AI agent is a system that interacts with its environment by perceiving inputs through sensors and acting upon it with actuators to interact as an output. It stores data and later on reasons about it to develop the best possible outcome to complete tasks in an autonomous way to achieve a goal set by a human. Learning from its experiences to improve future performance.

- **Simple Reflex Agents**

This type of agent works specifically with a set of well defined rules and immediate input data, they don't consider the past received information, which as its name can tell, we are talking about a very simple agent that doesn't require intensive training based on the condition-action relationship.

- **Goal Based Agents**

These agents have enhanced reasoning capabilities. They evaluate the information from the environment and also compare different approaches to achieve their goal by planning and choosing the most efficient approach. They also react to immediate changes in the environment while adapting and pursuing their objectives.

Environments
The environment defines the limits, rules and complexity level. Is everything that surrounds the AI agent and what said agent can use to interact, such as the inputs that change the state and how the agent must behave and it can also be the tool for the agent to execute outputs.

The environments can have different attributes, such as **accessibility** in the data, **determinism** in the outcomes, **episodicity** for how one action can affect the future ones, the **dynamics** of how the environment change with or without an agent´s action, the **discretization** to understand the number of possible states, the number of agents that interact with the said environment and finally the **knowledge** of the environment to understand the rules.

Development

Before discussing the implementation of a solution, we must first define the settings and rules of Battleship clearly and in detail. This step is necessary to properly program both the environment and the behavior of the AIs; for this reason, we dedicate a subsection to explaining everything that needs to be known about the game. The rules of Battleship are defined according to Official Game Rules (Official Game Rules, n.d.).

Explaining Battleship

Is a classic strategic board game where players compete to sink each other's fleet by guessing the locations of their opponent's ships on a grid.

Components

- **Game Boards:** Each player has a primary ocean grid (10×10) for positioning their fleet and a secondary targeting grid to track shots fired. It's only a two player game, where they compete against each other.
- **Ships:** Each player has a fleet of five ships:
    - Carrier (5 spaces)
    - Battleship (4 spaces)
    - Cruiser (3 spaces)
    - Submarine (3 spaces)
    - Destroyer (2 spaces)
- **Peg Markers:** Blue pegs to mark misses, and red pegs to mark hits.

Objective

Be the first player to sink all your opponents ships before they sink yours.

Setup: Positioning Ships:

- Each player places their five ships on their ocean grid. Ships can be placed either horizontally or vertically, but not diagonally.
- Ships must be placed within the boundaries of the grid and cannot overlap each other.
- Each player keeps their ship placement hidden from the other player.

Gameplay

Players take turns calling out coordinates to attack the opponent's ships, attempting to guess the locations of the ships.

1. **Calling a shot:**

- On a player's turn, they call out a coordinate.
- The opponent checks their ocean grid and announces whether the shot is a "hit" or a "miss".
- If it's a hit, the opponent places a red peg on their ship at that coordinate.
- If it's a miss, the opponent places a blue peg on their ocean grid at that coordinate.
- The player calling the shot marks their targeting grid with a red peg for a hit or a blue peg for a miss.

2. **Recording Hits and misses**

- Players keep track of their hits and misses on their targeting grid to help refine their strategy.
- Each time a ship is hit but not sunk, the player must say "hit" and specify which ship was hit once the ship is sunk

Sinking Ships:

- The opponent must announce which ship has been sunk.
- A ship is considered sunk when all of its coordinates have been hit.

Winning the Game

- The game continues in alternating turns until one player has sunk all five of their opponent's ships.
- The first player to sink all of their opponent's ships is the winner.

Tips and Strategies

- **Balanced Distribution:** Avoid clustering your ships too closely together to make it harder for your opponent to score consecutive hits.
- **Pattern Shots:** Use patterns like checkerboards or systematic sweeps to ensure thorough coverage of the grid.

- **Tracking Hits:** Pay attention to where your hits are and use logical deduction to target likely adjacent spaces to sink ships.

Flowchart of the game sequence

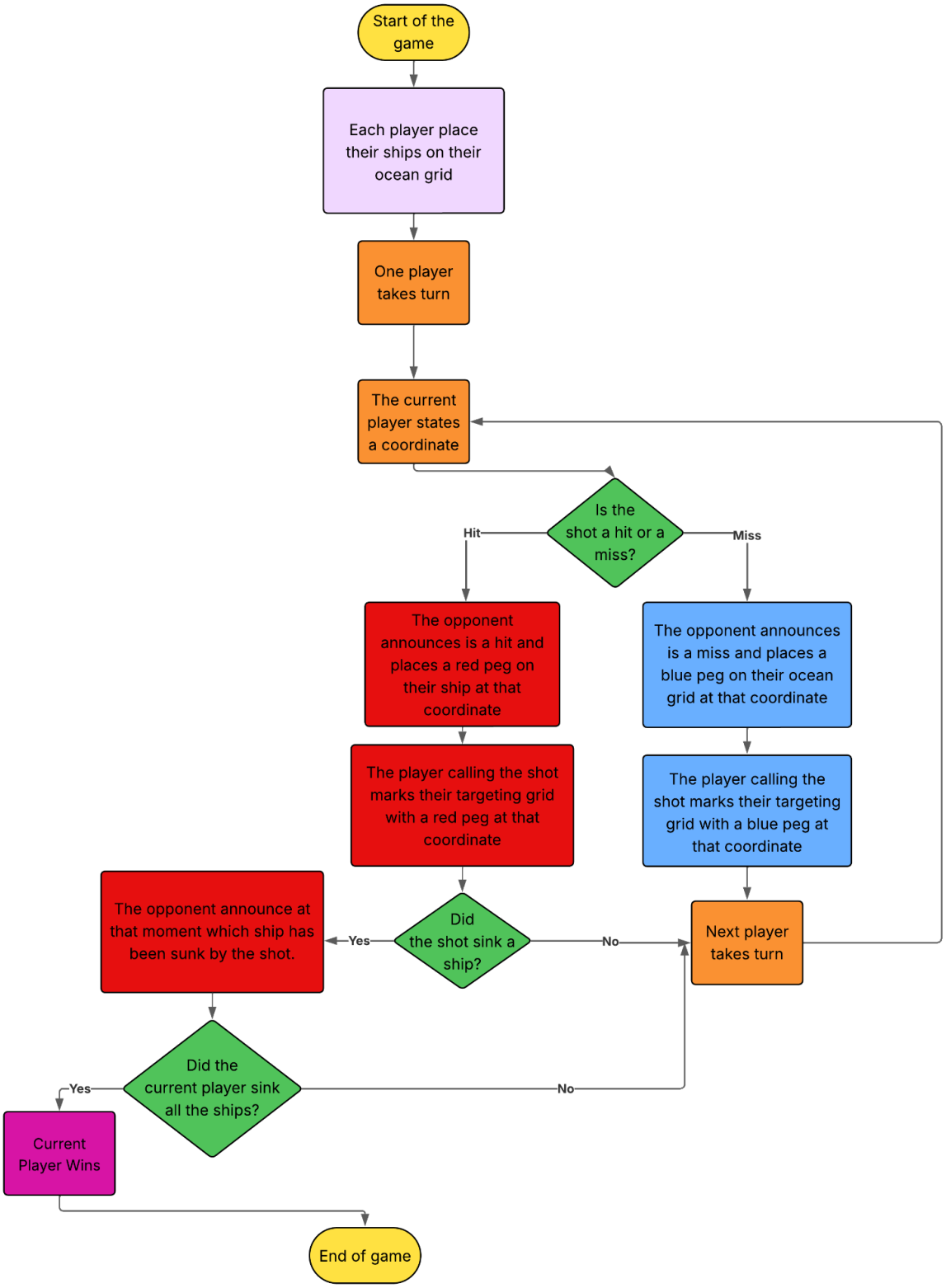

Chart 1. Flowchart of the sequence of the Battleship board game

Game implementation design

Having a complete layout of the game, now we can start talking about creating the program. According to our Goals, we have to design and implement a program that can play Battleship having three options for our two possible players: two AI agents of different nature and the user. We also have to design a simple Graphical User Interface (GUI) in order to present our program in class.

Developing the environment

We used an OOP (Object-Oriented Programming) paradigm to solve the problem, breaking a complex problem into small problems solved by different classes that interact with each other on a main function. First of all, the game requires a multiagent environment that can play against each other, in this case bots vs bots or a bot vs a human. Then we used a partially observable environment to hide the information about our boats position to the opponent as well as we don't know the opponent's boats information. Thus, revealing the information turn by turn in the board as in the original game, This makes the environment non-deterministic, since different outcomes may occur depending on the players previous actions. In which, a dynamic environment was just natural to show how the game evolves in real time, but with infinite states (continue).

GUI Basics

Having the main idea done, the way we planned to display the program and in general, the game, consists in a GUI (Graphic User Interface) that allows us first, to select in a menu which option to choose clicking the button in the interface, the options go from playing against the AI (Simple Reflex Agent or Goal-Based Agent), or emulate a game between AI's that will be explained further on. There's even the option to exit the program, giving the user the freedom of choosing what to do. This interface was made using the Tkinter library for python taking advantage of its simplicity in window design and easy incorporation of logic.

Playing against the AI (Simple or Goal-Based Agent)

In order to understand how the game fully works we first have to understand the way the game is presented to the user, and this is by drawing in the GUI two grids, the bigger one is the grid that represents the user's part of the board, while the smaller one represents the opponent's part of the board. Both can be interacted with and have a dimension of 10 columns and 10 rows, making a hundred different cells in a coordinate system ('A1' For example). Once the board is drawn the user has to place the boats on his own grid. In this step the game will tell the length of every boat the user is about to place, besides this, the player selects the orientation of the boat with a button placed below the board that allows him to

place it in a horizontal or vertical way. The placing goes from left to right (horizontal) and up to down (vertical). It's important to notice that the placing can’t be overlapping boats, every boat has to be on a not used cell in order to be placed correctly, once the user finishes placing their boats the AI begins its placing, this depending on which agent is our opponent:

The simple reflex agent does a random placement of boats, while the Goal-Based Agent strategically places the boats in cells that are statistically less possible to be at.

Once the user and the AI have positioned their boats, each fleet remains hidden from the opponent, starting the game with the user guessing the location of the opponent’s fleet, and by that ending his turn with either a hit or miss. This goes on playing while alternating turns until one side has sunk all of the other’s boats.

Simulation

The simulation is a panel that gives the user the possibility to look at the results from very quick games between both AI’s (Simple Reflex Agent vs Goal-Based Agent) the number of games simulated depends on a number the user inputs. This option is a performance test for the both agents in order to visualize how they act without user interaction.

Workload

The workload was divided into classes and functions, with each team member contributing to different parts of the code: from the logic, planning the paradigm, proposing the solutions to problems and finally testing. The OOP approach allowed us to clearly separate responsibilities, such as handling the game board, managing the players and how they were going to behave, from the way they placed their fleet, to how they tried to destroy the other player’s fleet. Also integrating the interface to give a final and clean look for the program. The design made collaboration efficient and made the documentation easier, since everyone understood the way the program works.

Results

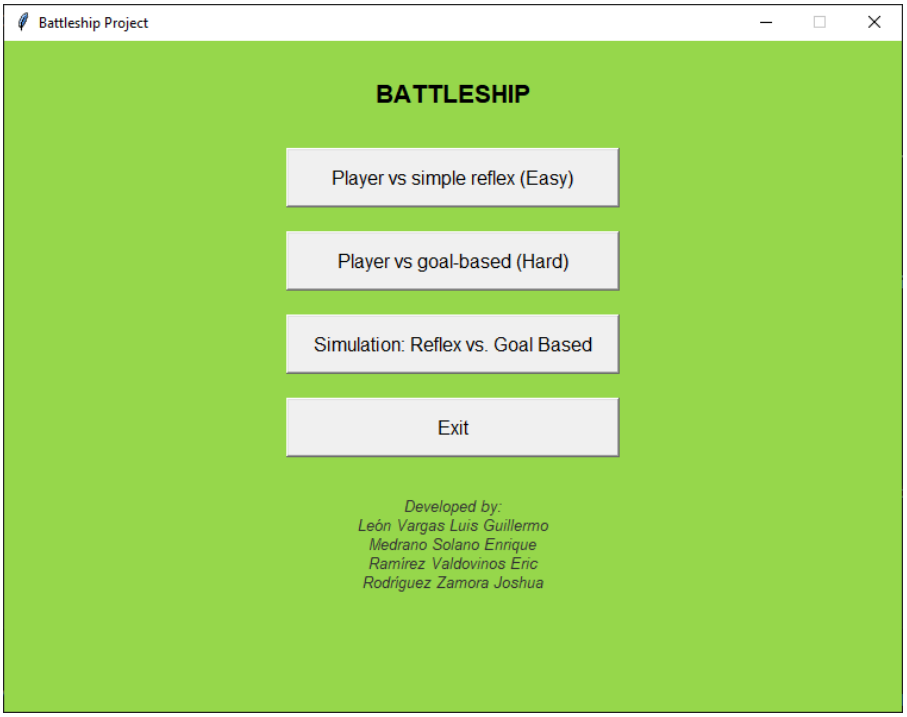

Figure 1. Main menu of the Battleship program

Figure 1 shows the main menu developed for the game. It provides the options required to meet the project's objectives: playing against a simple reflex agent, playing against a goal-based agent, or running a simulation between the two AIs for a specified number of games.

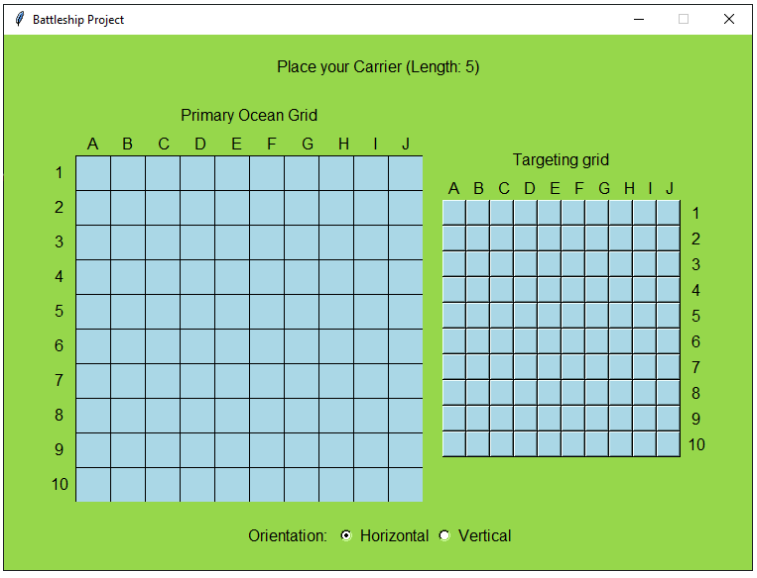

Figure 2. Placement phase when playing against an AI

After selecting an AI opponent, the game enters the placement phase, shown in ***Figure 2***. During this phase, the user places ships onto the grid, with the option to set a horizontal or vertical orientation for each one. As demonstrated in ***Figure 3***, the program validates each

placement, preventing a user from putting a ship outside the grid or in a location already occupied by another vessel.

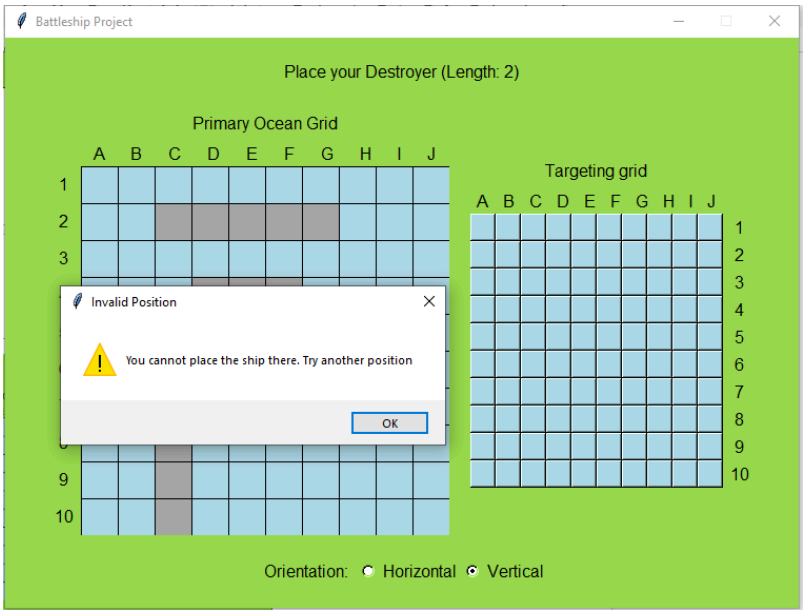

Figure 3. Error message when you try to place your boat outside the grid or into another boat

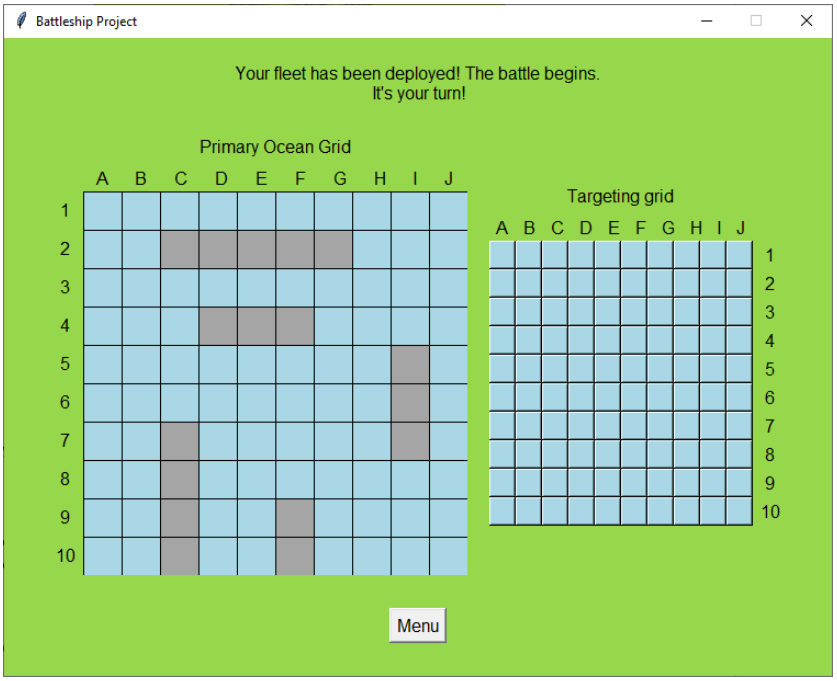

Figure 4. Ending of the colocation phase when playing against an AI

After the placement phase, combat begins, as shown in Figure 4. The user attacks by selecting coordinates on the opponent's grid. A successful hit on an enemy ship is marked in red, while a miss is marked in white. The status bar provides feedback on the coordinate attacked and the result. The AI takes its turn similarly, and the outcomes of its shots are

displayed on the user's primary grid. Figure 5 shows an early stage of combat, where both sides have started to land hits on each other's ships

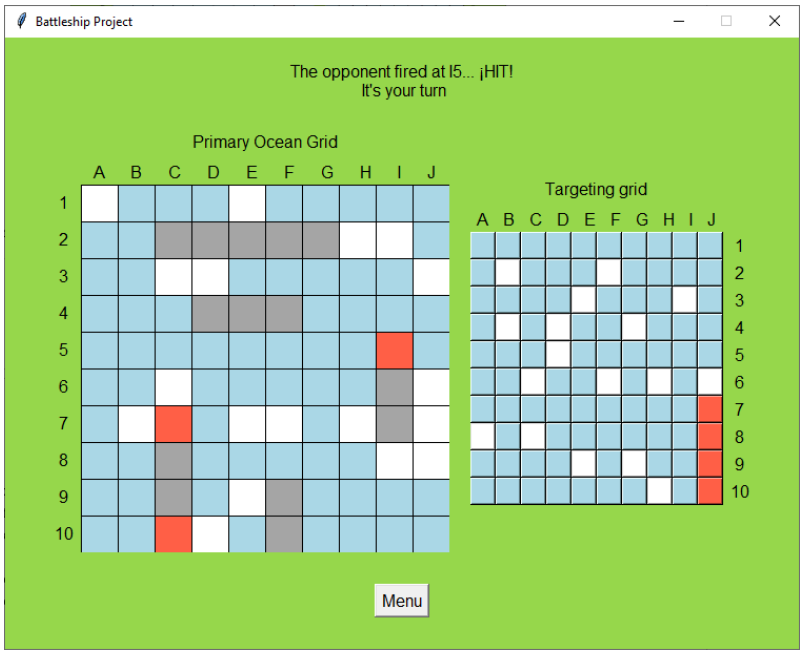

Figure 5. Playing against the Simple Reflex AI

During the initial ship placement phase, the gameplay experience is identical regardless of the AI opponent. The differences become clear once combat begins.

The simple reflex agent, shown in ***Figure 5***, employs a random search to locate the player's ships. When it scores a hit, its strategy is basic: it will attack an adjacent square only once. If that follow-up shot is a miss, the agent "forgets" the initial hit and reverts to random guessing. This forgiving behavior results in an easier experience, allowing the player a larger margin for error. ***Figure 6*** illustrates this, showing a scenario where the player was able to defeat the reflex agent despite making several inaccurate shots.

In contrast, the Goal-Based agent is a much less forgiving opponent. As seen in ***Figure 7***, it does not attack randomly but follows a calculated search pattern. Once it finds a vessel, it systematically hunts it down until it is sunk, as evidenced by the complete destruction of the ships it has targeted. This creates a much more challenging playthrough where careless errors can be quickly capitalized on by the AI. ***Figure 8*** demonstrates this, showing a game where the player was defeated after the Goal-Based agent gained an advantage.

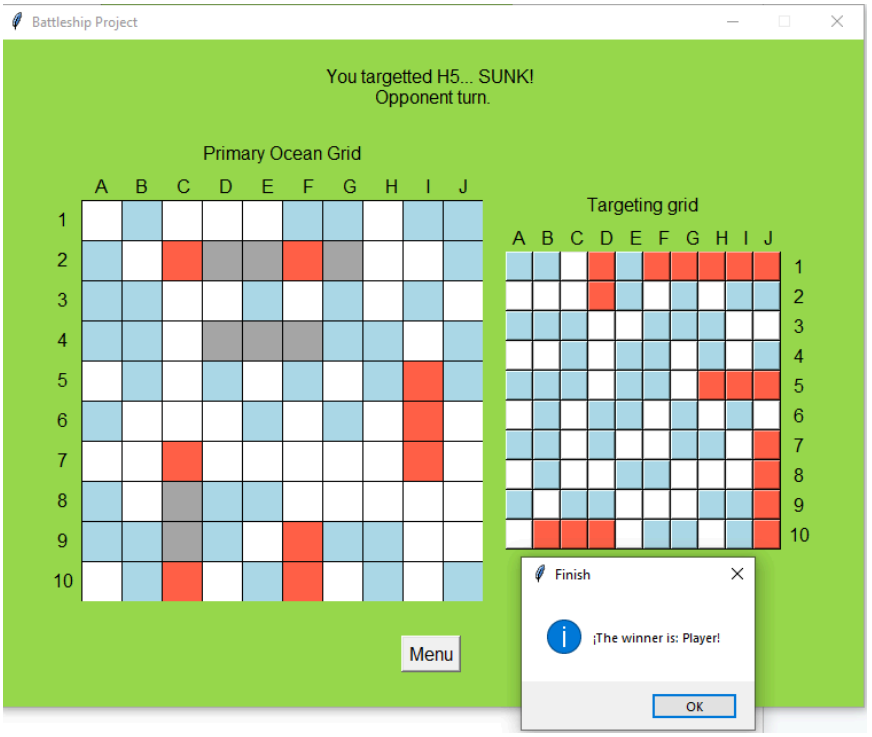

Figure 6. Result in playing against the simple reflex agent

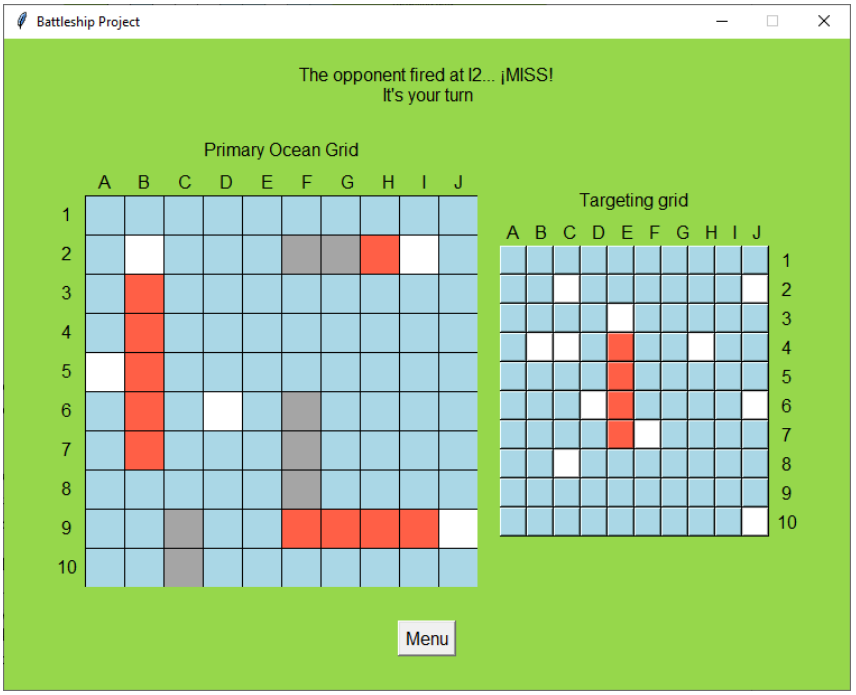

Figure 7. Playing against the Simple Reflex AI

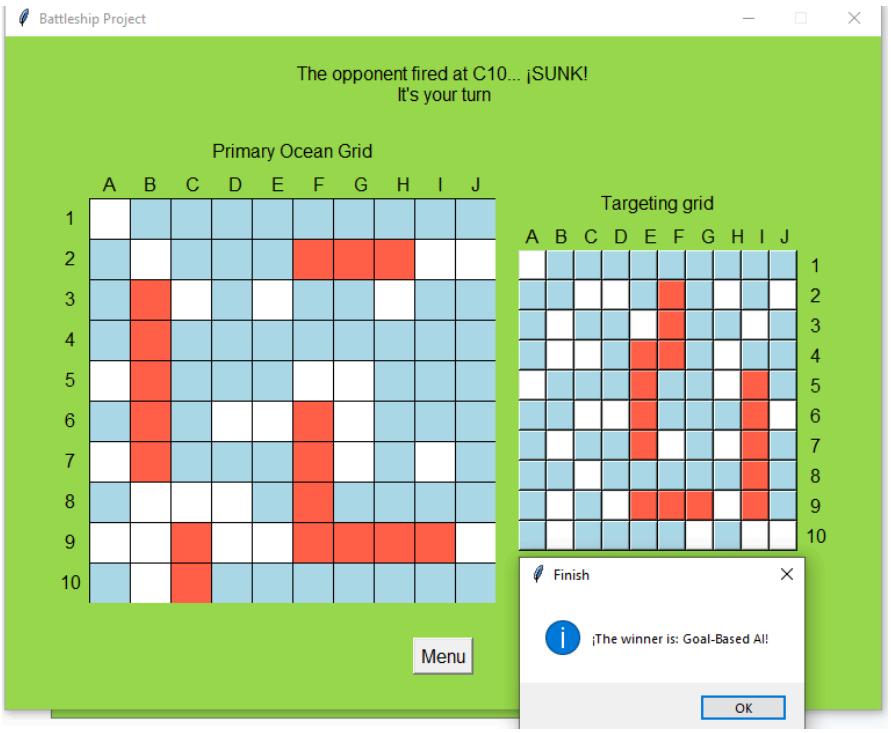

Figure 8. Result in playing against the simple reflex agent

The superiority of the Goal-Based agent over the Reflex agent becomes even more evident when the two are pitted against each other in a simulation. ***Figures 9*** through ***12*** present the statistics gathered from these automated matchups.

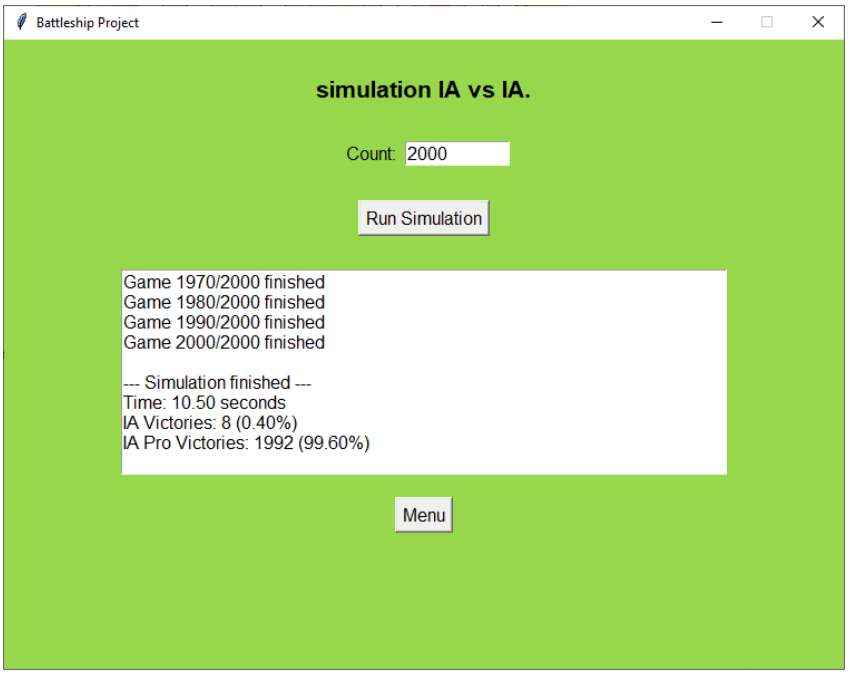

Figure 9. First set of automatic matchups in the simulation

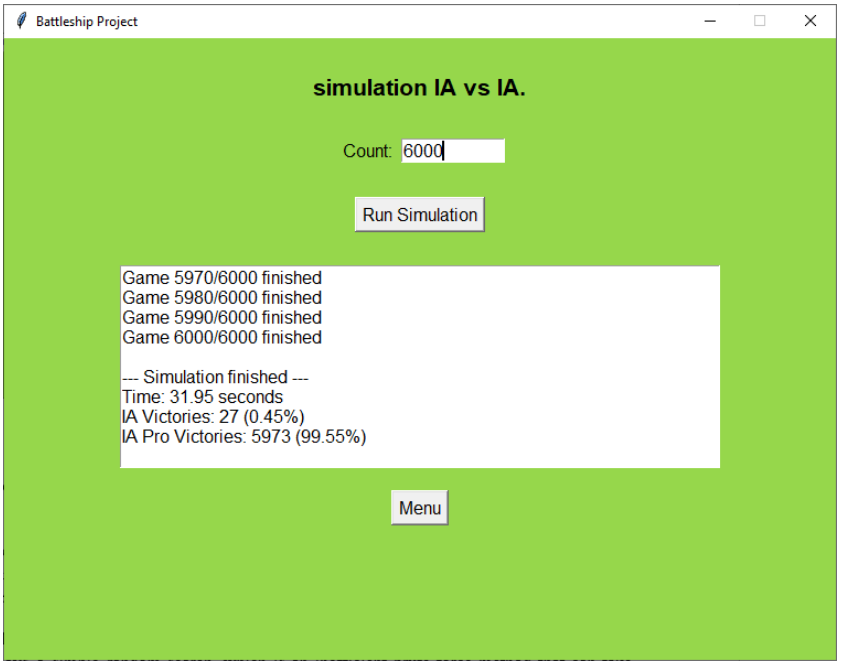

Figure 10. Second set of automatic matchups in the simulation

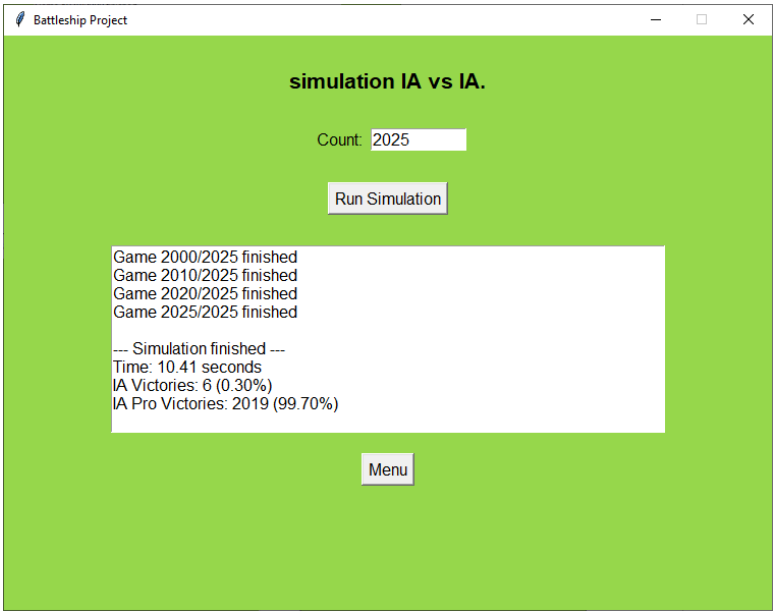

Figure 11. Third set of automatic matchups in the simulation

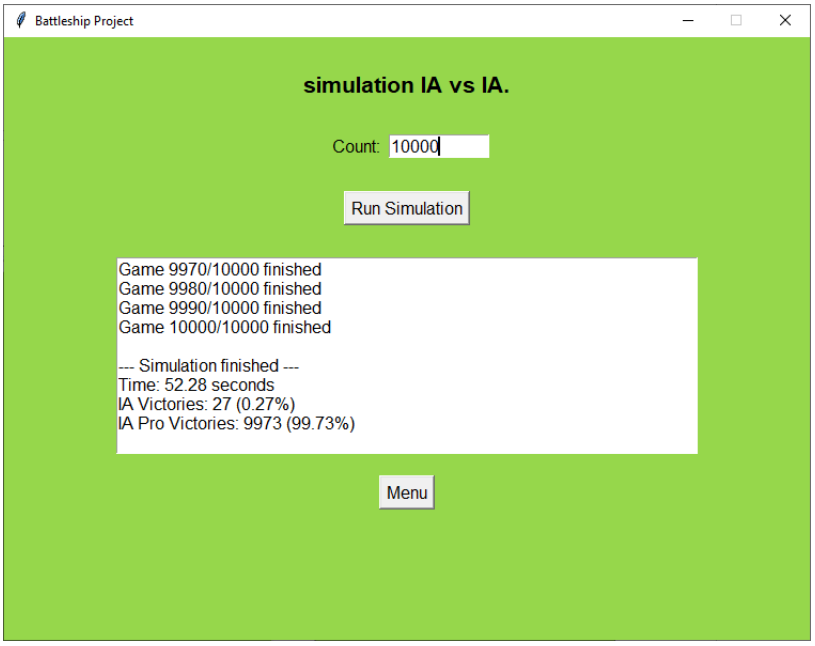

Figure 12. Fourth set of automatic matchups in the simulation

In all matchups, the Goal-Based agent overwhelmingly defeated the reflex agent, winning more than 99.5% of the games. The consistent victories of the goal-based agent over the reflex agent are not due to chance, but are a direct result of superior algorithmic efficiency in the two critical phases of the game: searching for ships and destroying them.

Firstly, their search strategies are fundamentally different in effectiveness. The reflex agent employs a simple random search, which is an inefficient brute-force method that can take many turns to locate an enemy vessel. In contrast, the goal-based agent uses a systematic checkerboard pattern. This optimized approach guarantees it will find any ship by targeting, at most, only half the board, giving it a significant head start in every game.

Secondly, their sinking strategies highlight the importance of memory and logic. The reflex agent's memory is limited to its single last hit; if an adjacent shot misses, it abandons the known target and reverts to random guessing, wasting valuable turns. The goal-based agent, however, enters a dedicated "target" mode, remembers all hits on a specific ship, and deduces the ship's orientation. By logically targeting the ends of a discovered vessel, it ensures a swift and efficient kill with minimal wasted shots.

Conclusions

It can be concluded that the development of the Battleship project tested the diverse skills of all team members, not only in programming but also in designs, documentation , and the logical reasoning required to solve the problems that arose.

Reflex Agent was developed with simple game logic and limited memory of its past moves. The Goal Based Agent features an improved algorithm that allows it to remember and store

previously made moves, giving it a strategic advantage. This design allowed the user to choose which type of AI to compete against.

Additionally, a simulation was run between both agents (Reflex Agent vs Goal Based Agent) to measure their performance. The results, which showed the number of games won by each AI and their respective win percentages, lead to the conclusion that the Goal Based Agent has a clear superiority. This is due not only to its ability to store information about shots during the game but also to a more intelligent ship placement strategy at the start, in contrast to the Reflex Agent's random placement.It's way better for two main reasons: it remembers where shots landed during the game, and its initial ship placement is smarter, not just random like the Reflex Agent's.

Finally, the entire project was integrated into a graphical user interface developed with Python's Tkinter library, the same language used to implement all the programming logic. This resulted in a more straightforward and intuitive interaction for the user, both for playing against the AI agents and for running the simulation.

Bibliography

- Russell, S. J., & Norvig, P. (2016). Artificial Intelligence: A modern approach (3rd ed., Global ed.). Pearson.
- How to Play Battleship | Official Game Rules. (2025, March 13). Official Game Rules. https://officialgamerules.org/game-rules/battleship/
- Brainpod AI. (2025, marzo 16). *Explorando la IA racional comprendiendo la Racionalidad, los Agentes y su Papel en los Sistemas Inteligentes.* https://brainpod.ai/es/explorando-la-inteligencia-artificial-racional-entendiendo-la-racionalidad-de-los-agentes-y-su-papel-en-los-sistemas-inteligentes/#
- *¿Qué son los agentes de inteligencia artificial?: explicación de los agentes de inteligencia artificial - AWS. Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/ai-agents/*