EMAI-611 Course Project Final Report

Battleship Game using EasyAI (Python)

King Abdul Aziz University

Faculty of Computing and Information Technology

Computer Science Department

EMAI-611: Advanced Programming for AI

By:

Ghada Alsulami

Raseel Alghamdi-

May 18, 2025

Table of Contents

Introduction

This project aims to design and implement a Battleship game using the Python programming language, integrating an AI opponent using the easyAI library. Battleship is a two-player strategy game where each player positions a set of ships on a hidden grid and attempts to sink the opponent's fleet through strategic guessing. This implementation allows a human player to compete against an AI that uses the Minimax algorithm.

Incorporating AI into game development not only enriches the user experience but also demonstrates practical applications of AI decision-making techniques. This project highlights how classical game logic can be combined with modern AI tools to create engaging and interactive software.

Game Development Setup and Implementation

The Battleship game was built in Python within a Jupyter Notebook, utilizing the `easyAI` library for AI decision-making. Development was done in Google Colab.

Programming Environment:
- Python 3.x
- Libraries used: `easyAI`, `random`

AI Algorithm:
- The AI opponent uses the Minimax algorithm, implemented via easyAI's `AI_Player` class. This enables the AI to make optimal moves based on the game state.

Game Design Features:
- A 6x6 grid (A–F columns, 1–6 rows)
- Ship sizes: 2 and 3 units
- Players take turns guessing grid coordinates
- AI evaluates and chooses the best move based on game state

Key Implementation Details:
1. `BattleshipGame` class extends `TwoPlayersGame` from easyAI.
2. Player boards are initialized with ships placed randomly.
3. AI move logic is handled through `possible_moves()` and `make_move()`.
4. Game ends when all of one player's ships are sunk.

Table 1: A diagram showing the 6x6 board with sample ship placements

	A	B	C	D	E	F
1	.	.	.	.	.	.
2	.	.	S	.	.	.
3	.	.	S	.	.	.
4	.	.	.	.	S	S
5	.	.	.	.	.	.
6	.	.	.	.	.	.

Results

The implementation was successful and the AI performs strategically. The turn-based system works correctly and detects winning conditions accurately. The player can interact via the console, and the game board is printed after every turn, displaying hits, misses, and remaining ships.

Implemented Features:
- Functional Battleship gameplay
- AI vs Human interaction using turn-based input
- Hit/miss feedback with visual grid updates
- End game detection and scoring logic

Table 2: Game Features and Implementation Status

Feature	**Implemented**	**Description**
AI Move Logic	Yes	Minimax algorithm decision making
Ship Placement	Yes	Randomized for both players
Win/Loss Detection	Yes	Ends game when all ships are sunk
Input Validation	Partially	Basic checks for valid input

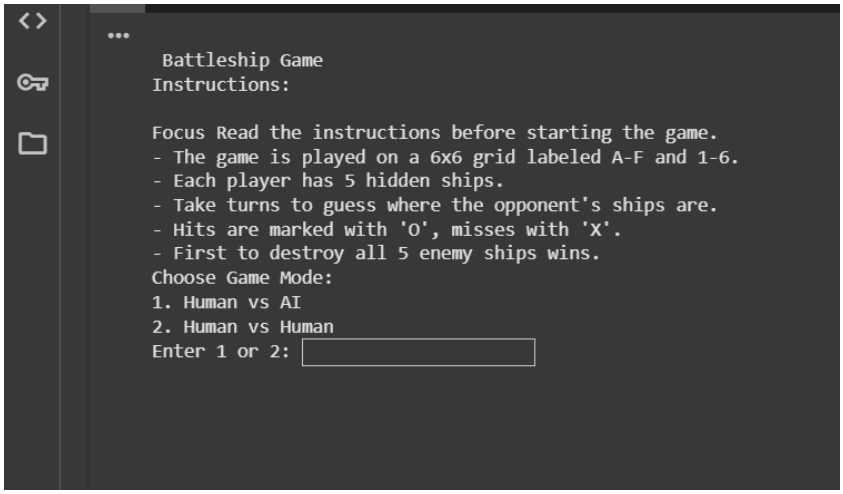

Figure 1: screenshot of the beginning of the game, with the instructions, modes

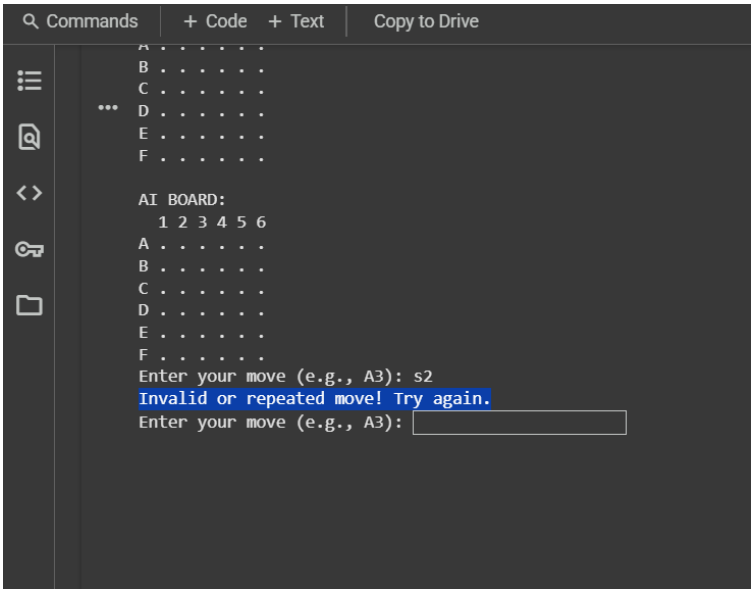

Figure 2: screenshot of when you enter a wrong index or letter that is not in the range

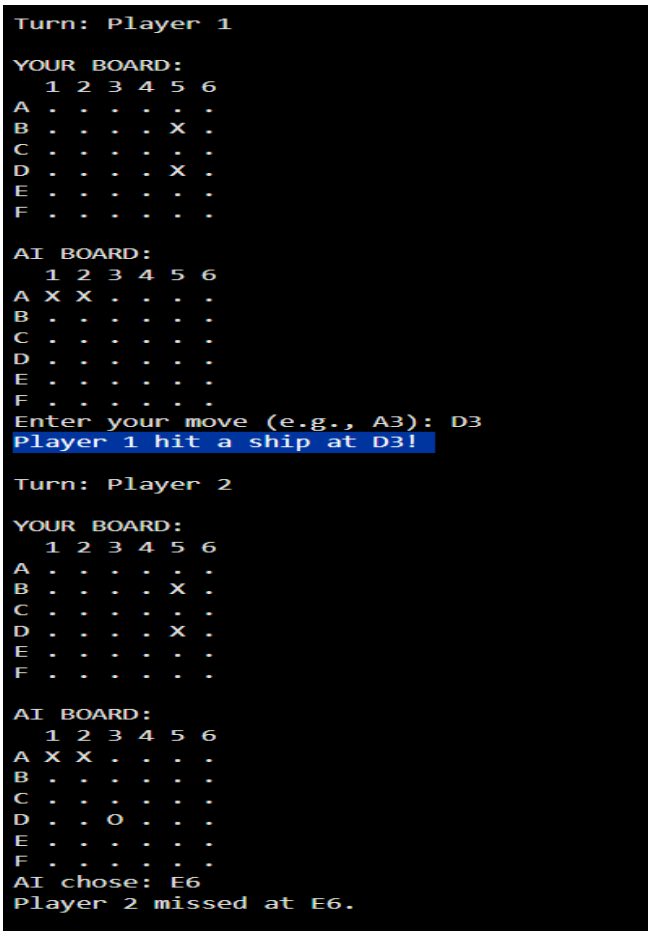

Figure 3: screenshot of a sample game session (board with hits and misses), hit = O, Miss = X

Table 3: Table showing sample input and output per turn

Turn	Player	Guess	Result
1	AI	B3	Miss
2	Human	A1	Hit
3	AI	A2	Hit
4	Human	C3	Miss

Challenges

Several technical and conceptual challenges were encountered:

- Adapting `easyAI` to fit the unique Battleship game flow (which differs from games like Tic-Tac-Toe)
- Managing dynamic grid updates and tracking of multiple ship positions
- Ensuring fair AI ship placement and legal moves
- Handling user input and error validation for coordinate guesses

These challenges were tackled through modular code design, frequent debugging, and testing AI behavior under various board states.

Conclusion and Future Work

This project showcased the integration of artificial intelligence into a classic strategy game using Python. By leveraging the easyAI library, the AI is capable of making intelligent moves that challenge the human player. The implementation was kept modular and testable.

Planned Improvements:
- Add a graphical interface using libraries like Tkinter or PyGame
- Enable custom ship setup by the user
- Support multiplayer over a local network or the internet
- Improve the AI to consider player guess patterns

Here are some screenshots for future work and Improvements, we have added a graphical interface using libraries like Tkinter or PyGame and enabled custom ship setup by the user

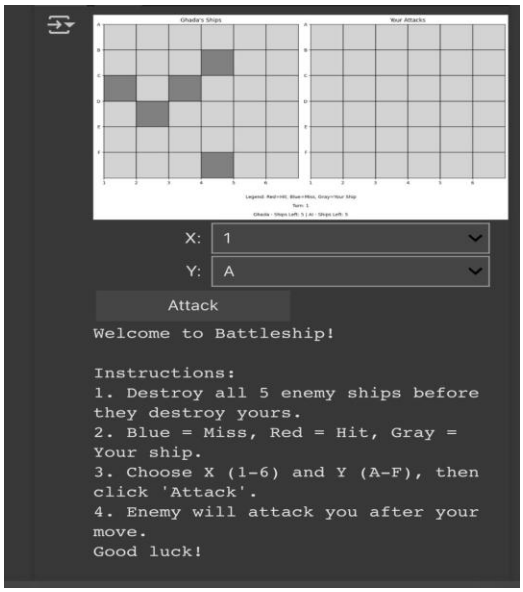

Figure 4 : Here are the boards for player 1 and player 2

Figure 5 : Here you can play with Arabic and English

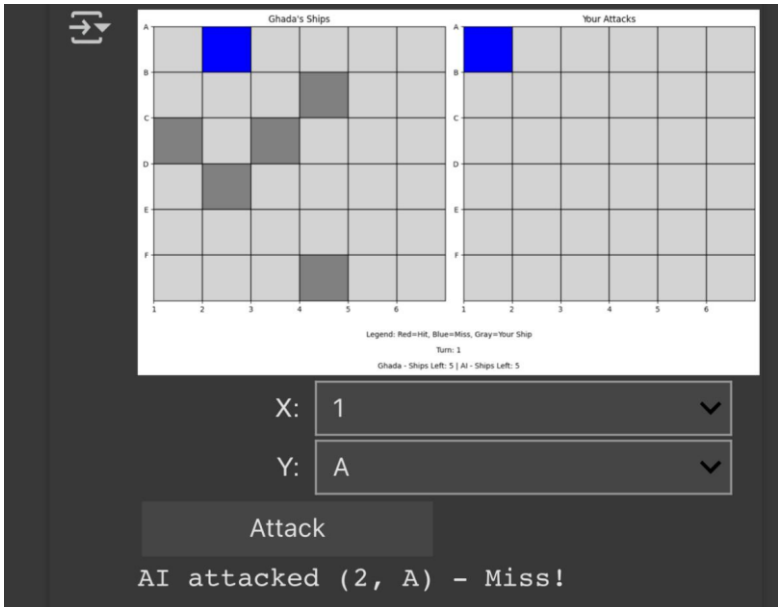

Figure 6 : Here is the output for choosing the cell you want, and it shows if it hit or miss

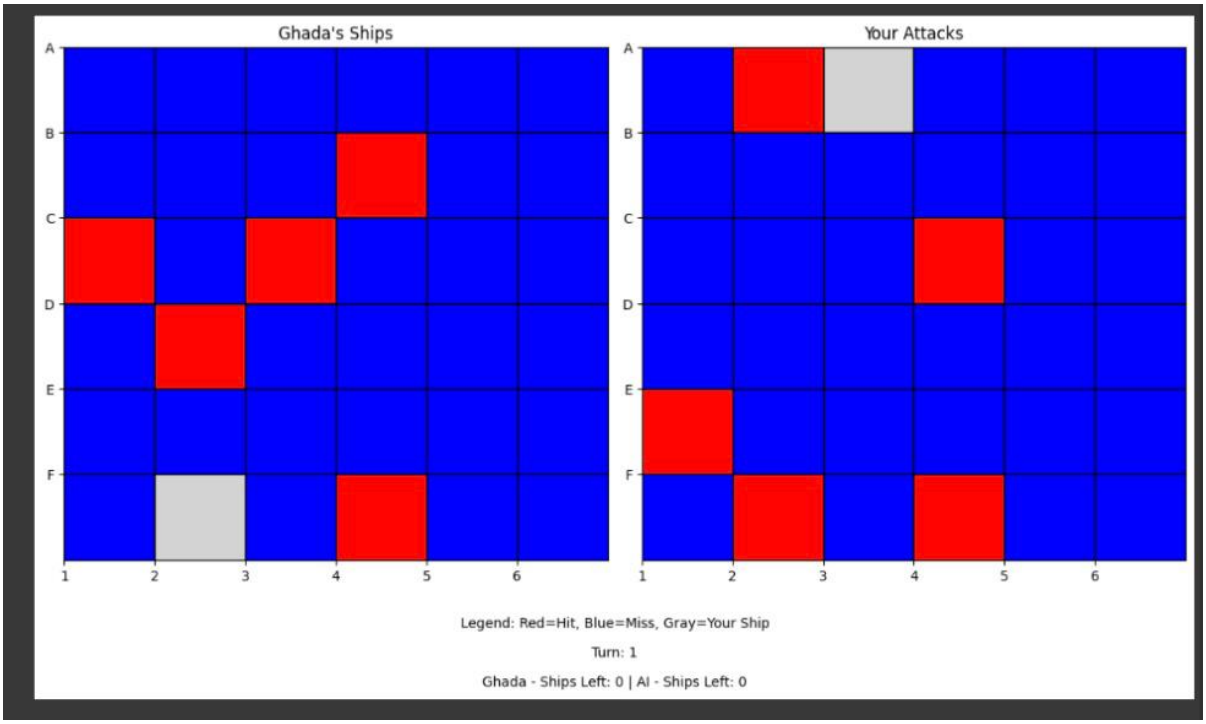

Figure 7 : Here is the interface, it shows the players' boards, if red = hit, blue = miss, grey = your ship, and in the bottom it shows the turn, and how many ships are left for the both players

Bibliography

- Zulko, easyAI documentation: https://zulko.github.io/easyAI/
- Python Official Docs: https://docs.python.org/3/
- EMAI611 Course Slides
- AI Game Programming Patterns by Robert Nystrom
- libraries like Tkinter or PyGame for the future work and improvements