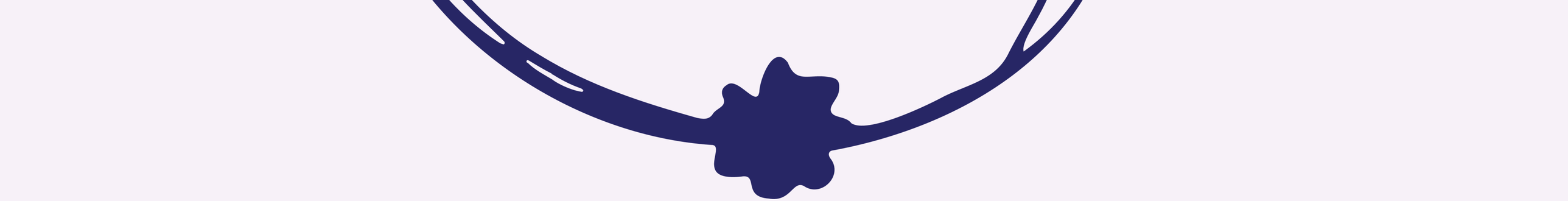
Battleship Game using easyAI (Python)

EMAI-611: Advanced Programming for AI

Raseel Alghamdi - [redacted]
Ghadah Alsulami - [redacted]



Introduction

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

.Classic Battleship game developed using Python and easyAI•

.Integrates AI opponent using Minimax algorithm•

.Demonstrates AI decision-making in a strategic board game•

Game Development Setup

.Python 3.x in Google Colab Notebook•

.Key Libraries: easyAI, random•

.Board: 6x6 grid (A-F columns, 1-6 rows)•

.Ships: Lengths 2 and 3•

AI Algorithm

01

AI uses easyAI's Minimax• .implementation

02

Evaluates possible moves and• .outcomes

03

Chooses optimal move each• .turn

Key Implementation

Game logic in BattleshipGame•
.class

AI_Player used to simulate•
.competitive gameplay

,Methods: possible_moves•
.make_move, lose, show

Class Definition Snippet

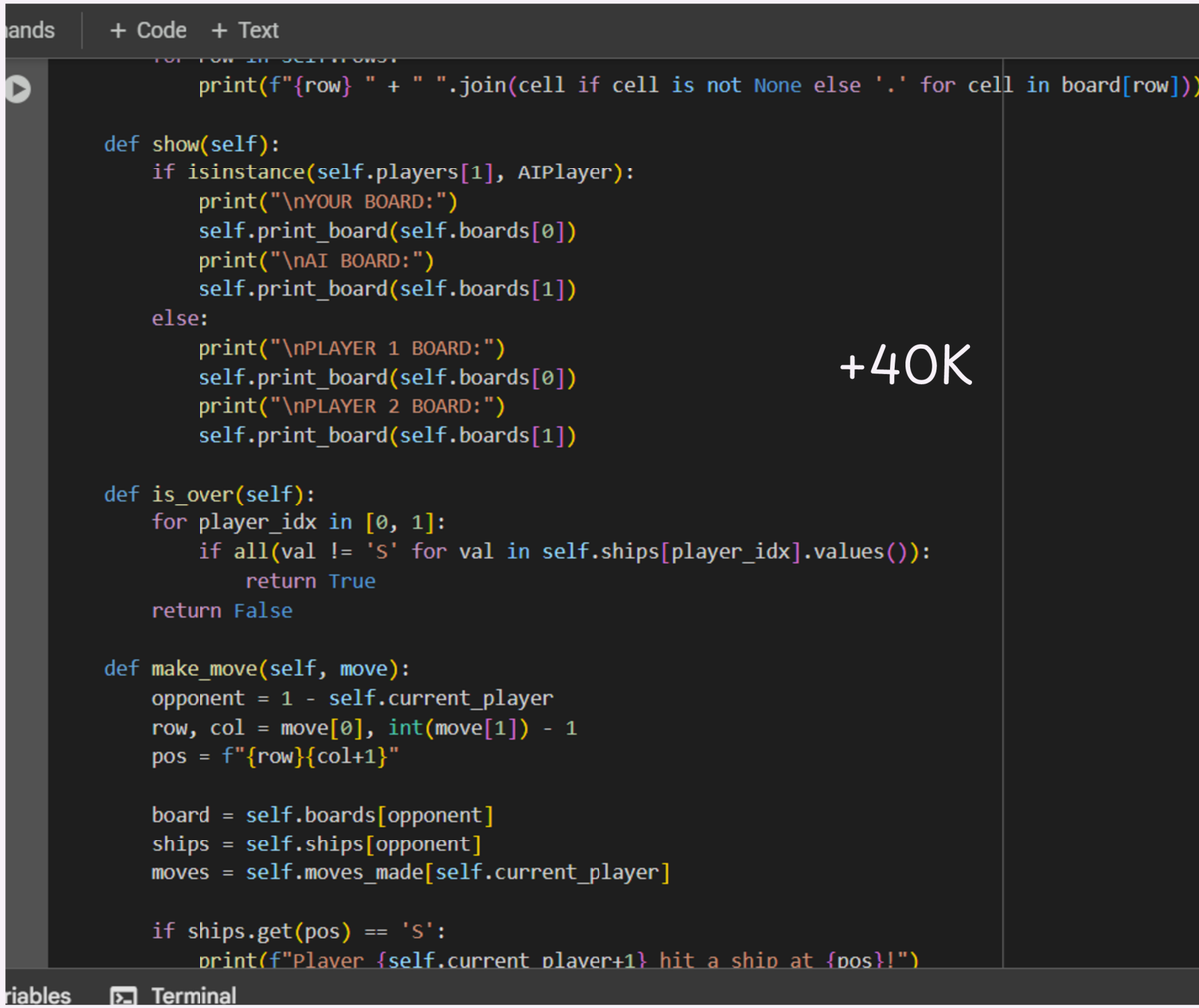
```python
import random

class BattleshipGame:
    def __init__(self, players):
        self.players = players
        self.board_size = 6
        self.rows = 'ABCDEF'
        self.boards = [self.init_board(), self.init_board()]  # Player 1 and Player 2 boards
        self.ships = [self.place_ships(), self.place_ships()]  # Ships for each player
        self.moves_made = [set(), set()]  # To track previous moves
        self.current_player = 0  # 0 for player 1, 1 for player 2

    def init_board(self):
        return {row: [None]*self.board_size for row in self.rows}

    def place_ships(self):
        ships = {}
        while len(ships) < 5:
            row = random.choice(self.rows)
            col = random.randint(1, self.board_size)
            pos = f"{row}{col}"
            if pos not in ships:
                ships[pos] = 'S'
        return ships

    def print_board(self, board):
        print("  " + " ".join(str(i+1) for i in range(self.board_size)))
        for row in self.rows:
            print(f"{row} " + " ".join(cell if cell is not None else '.' for cell in board[row]))
```



+40K

Class Definition Snippet

```
    def next_move(self):
        current = self.players[self.current_player]
        if isinstance(current, HumanPlayer):
            move = input("Enter your move (e.g., A3): ").upper()
            while not self.is_valid_move(move):
                print("Invalid or repeated move! Try again.")
                move = input("Enter your move (e.g., A3): ").upper()
        else:
            move = current.play(self)
            while not self.is_valid_move(move):
                move = current.play(self)
            print(f"AI chose: {move}")
        return move

    def play(self):
        print("\n Battleship Showdown")
        print("Try to sink all enemy ships before they sink yours!")

        while not self.is_over():
            print(f"\nTurn: Player {self.current_player + 1}")
            self.show()
            move = self.next_move()
            self.make_move(move)
            self.current_player = 1 - self.current_player

        print("\nGame Over!")
        winner = 2 if self.current_player == 0 else 1
        print(f"Player {winner} wins!")

class HumanPlayer:
    def play(self, game):
```

```
        if ships.get(pos) == 'S':
            print(f"Player {self.current_player+1} hit a ship at {pos}!")
            ships[pos] = 'H'
            board[row][col] = 'O'
        else:
            print(f"Player {self.current_player+1} missed at {pos}.")
            board[row][col] = 'X'

        moves.add(pos)

    def is_valid_move(self, move):
        if len(move) not in [2, 3]:
            return False
        row = move[0].upper()
        col = move[1:]
        if row not in self.rows or not col.isdigit():
            return False
        col = int(col)
        if not (1 <= col <= self.board_size):
            return False
        pos = f"{row}{col}"
        return pos not in self.moves_made[self.current_player]

    def next_move(self):
        current = self.players[self.current_player]
        if isinstance(current, HumanPlayer):
            move = input("Enter your move (e.g., A3): ").upper()
            while not self.is_valid_move(move):
                print("Invalid or repeated move! Try again.")
                move = input("Enter your move (e.g., A3): ").upper()
```

Terminal

Class Definition Snippet

```python
class HumanPlayer:
    def play(self, game):
        return input("Your move (e.g., A1): ").upper()

class AIPlayer:
    def play(self, game):
        row = random.choice(game.rows)
        col = random.randint(1, game.board_size)
        return f"{row}{col}"
```

```python
def run_game():
    print("\n Battleship Game")
    print("Instructions: ")
    print("\nFocus Read the instructions before starting the game.")
    print("- The game is played on a 6x6 grid labeled A-F and 1-6.")
    print("- Each player has 5 hidden ships.")
    print("- Take turns to guess where the opponent's ships are.")
    print("- Hits are marked with 'O', misses with 'X'.")
    print("- First to destroy all 5 enemy ships wins.")

    print("Choose Game Mode:")
    print("1. Human vs AI")
    print("2. Human vs Human")

    mode = input("Enter 1 or 2: ")
    while mode not in ['1', '2']:
        mode = input("Invalid input. Enter 1 or 2: ")

    if mode == '1':
        players = [HumanPlayer(), AIPlayer()]
        print("Mode: Human vs AI")
    else:
        players = [HumanPlayer(), HumanPlayer()]
        print("Mode: Human vs Human")

    game = BattleshipGame(players)
    game.play()

# Run the game
run_game()
```

Results

01

:Turn-based gameplay•

.Human vs AI

Human vs. Human

02

,Features: Ship placement•

hit/miss logic, win/loss

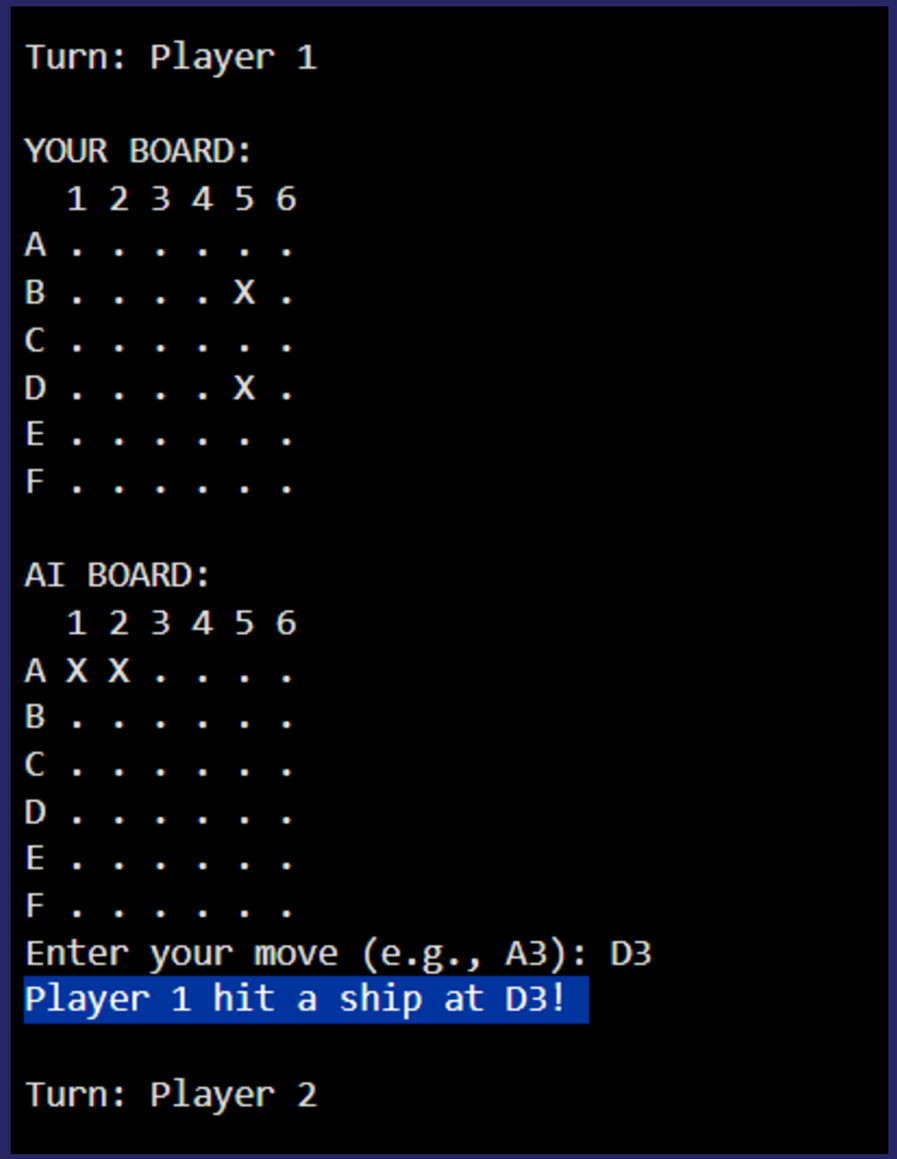
.detection

03

Outputs shown in console•

.after each move

Sample Game Output Screenshot



Game Features Table

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

Turn-by-Turn Gameplay Table

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

CHALLENGES

.Adapting easyAI for Battleship mechanics•

.Ensuring valid ship placements•

.Managing grid updates and player input•

.Handling turn logic and game state•

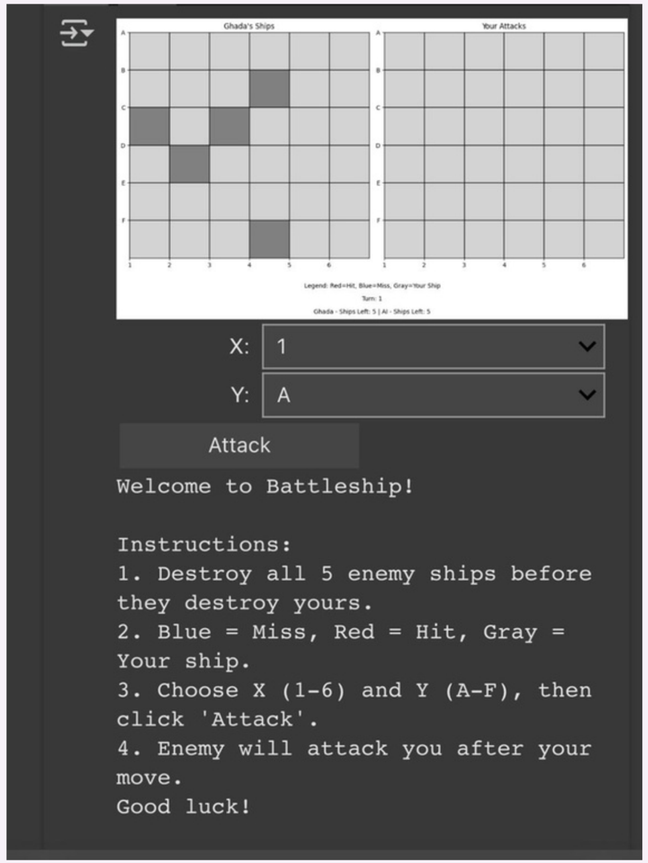
Conclusion & Future Work

.Project demonstrates AI integration in classic game design•

.Future improvements: GUI, custom ship setup, multiplayer, smarter AI•

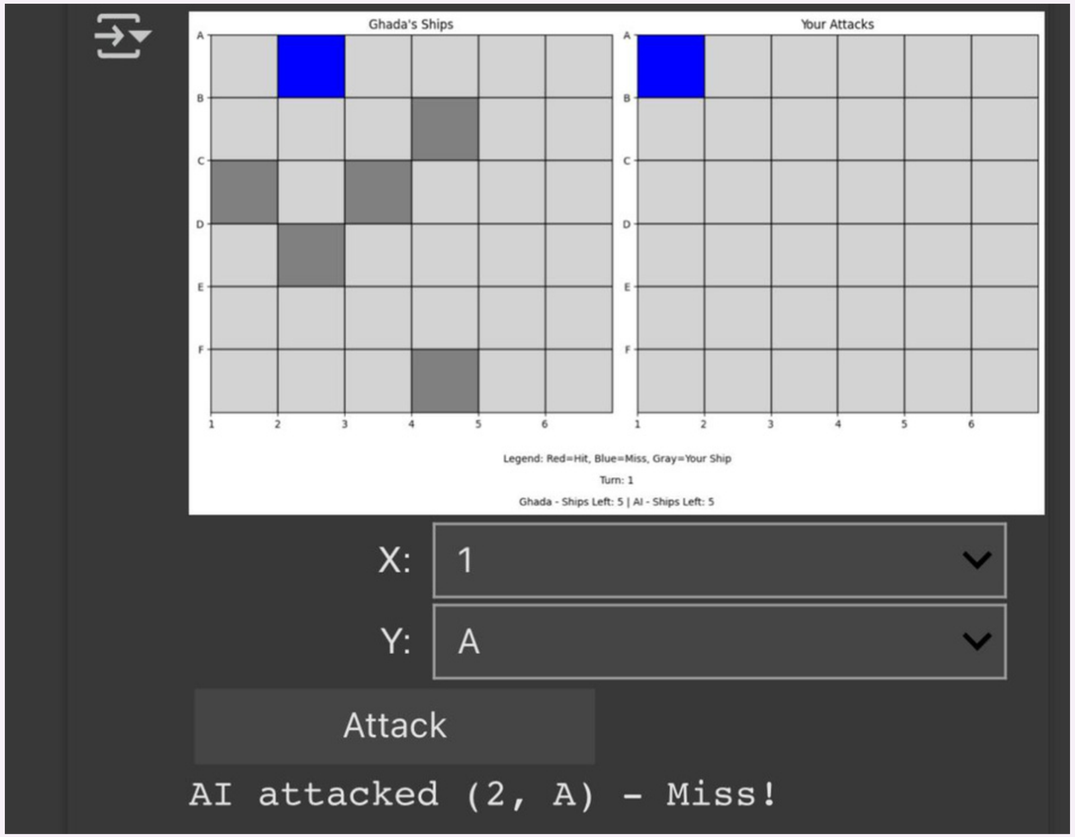
Here are some screenshots for future work and Improvements

Here you can play with Arabic and English



Here are the boards for player 1 and player 2

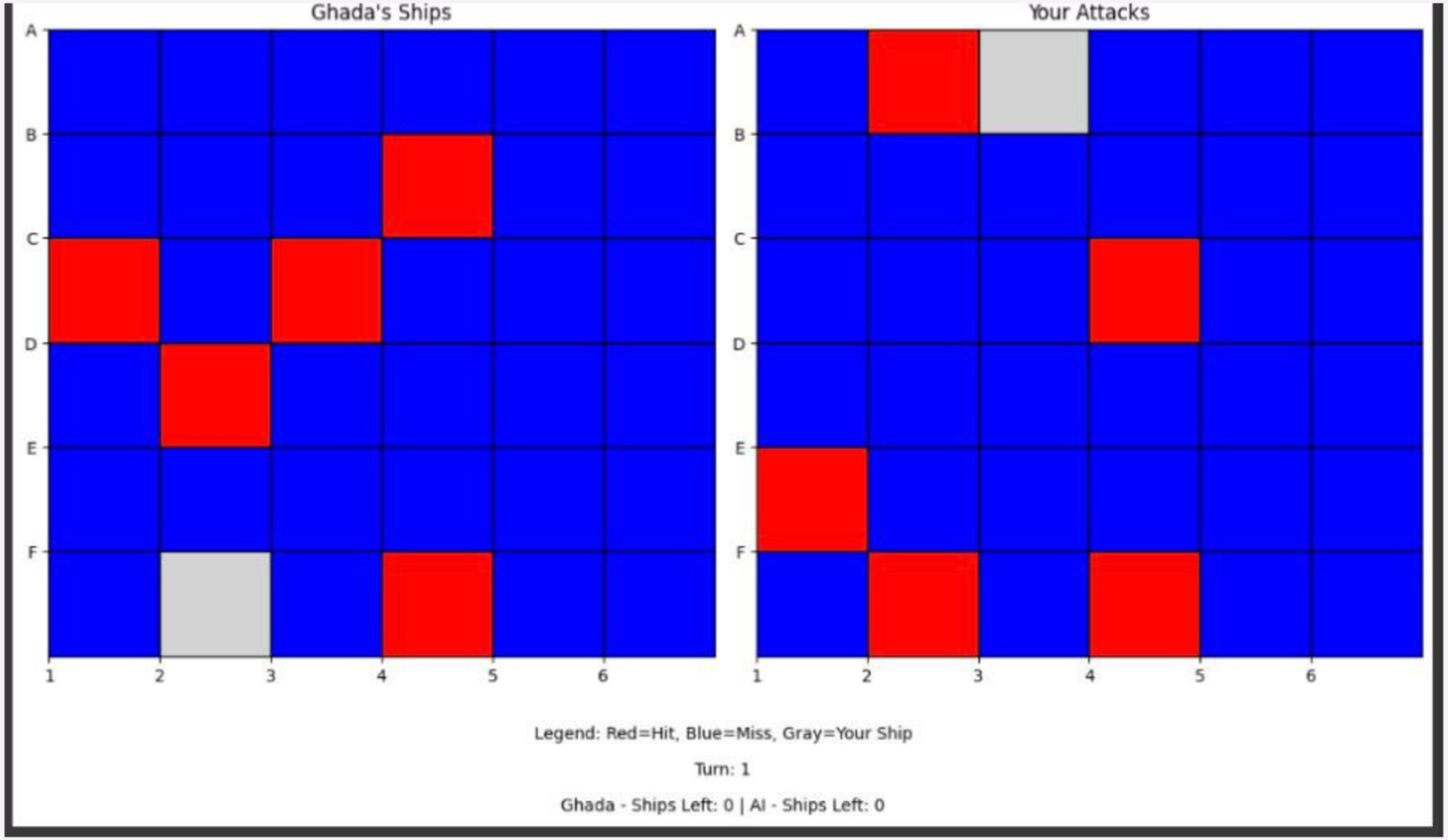
Here are some screenshots for future work and Improvements



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

Here are some screenshots for future work and Improvements

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

Thank you

for your listening! :)