CSE4110: Artificial Intelligence Laboratory

DoRa Block Battle

By

Md. Raduan Islam Rian

Roll: 1907117

&

Doniel Tripura

Roll: 1907121

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Khulna 9203, Bangladesh

September, 2024

DoRa Block Battle

By

Md. Raduan Islam Rian

Roll: 1907117

&

Doniel Tripura

Roll: 1907121

Submitted To:

Md. Shahidul Salim

Lecturer

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Most. Kaniz Fatema Isha

Lecturer

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Khulna 9203, Bangladesh

September, 2024

Acknowledgement

All praise is due to the Almighty God, whose blessings and mercy have guided me to successfully complete this coursework. We would like to express our sincere gratitude to our course teachers, Md. Shahidul Salim sir & Most. Kaniz Fatema Isha mam for their continuous support and guidance throughout this project.

Author

Contents

List of Tables

List of Figures

CHAPTER I

Introduction

1.1 Introduction

In the field of artificial intelligence (AI) and interactive systems, the development of intelligent game applications represents a significant advancement. This project focuses on creating a game application using the Tkinter library in Python, integrating sophisticated AI techniques to enhance the gameplay experience. By leveraging various AI algorithms, including alpha-beta pruning, genetic algorithms, fuzzy logic, and A* search, this project aims to demonstrate the practical application of these methods in a game setting.

The primary motivation behind this endeavor is to explore how AI can be utilized to develop games that are not only engaging and interactive but also demonstrate complex decision-making capabilities. Through the implementation of these algorithms, the project seeks to provide a compelling example of AI integration in game development, showcasing the potential for creating dynamic and responsive gaming experiences.

1.2 Objectives

The key objectives of this project are:

- **Develop a Tkinter-Based Game Application**: Create a user-friendly game interface using Tkinter, a widely-used library for building desktop applications in Python.
- **Integrate Various AI Techniques**: Implement multiple AI algorithms, such as alpha-beta pruning, genetic algorithms, fuzzy logic, and A* search, to enhance the game's intelligence and strategic depth.
- **Ensure Interactive and Dynamic Gameplay**: Design the game to support both single-player and multi-player modes, ensuring an engaging experience through adaptive and challenging AI.

- **Enhance Responsiveness and Usability**: Ensure that the game application is responsive to different screen sizes and provides a seamless user experience.
- **Provide Comprehensive Instructions**: Include an instructions page with detailed guidance on gameplay and the use of various AI techniques to assist users in navigating the game.

CHAPTER II

Theory

2.1 MinMax Algorithm

The MinMax algorithm is a decision-making algorithm used in two-player games to determine the optimal move. It is based on the principle of minimizing the possible maximum loss. In this algorithm, one player is assumed to be trying to maximize their score (Max), while the opponent is trying to minimize the score (Min). The algorithm recursively explores all possible moves, evaluating the resulting game states, and selects the move that leads to the optimal outcome for the Max player while assuming the opponent will also play optimally.

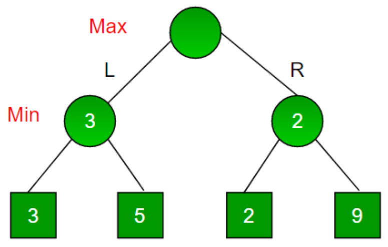


Figure 2.1: Minmax Algorithm

2.2 Alpha-Beta Pruning

Alpha-Beta Pruning is an optimization technique for the MinMax algorithm. It reduces the number of nodes evaluated in the search tree by pruning branches that cannot influence the final decision. Alpha represents the best value that the maximizing player can guarantee so far, while Beta represents the best value that the minimizing player can guarantee. During the search, if a node's value is found to be worse than the current Alpha or Beta values, further exploration of that node is cut off, thus improving efficiency.

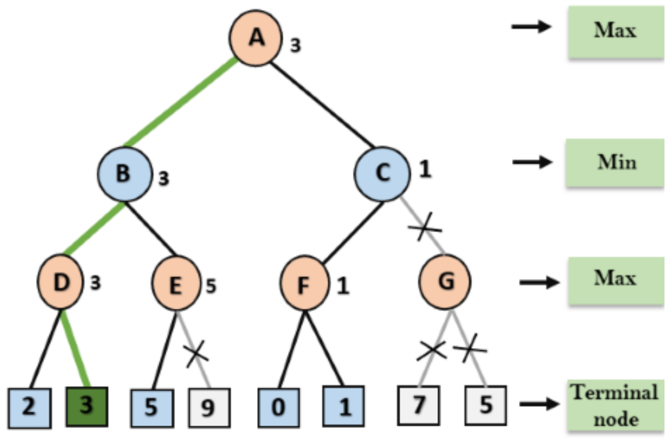


Figure 2.2: Alpha Beta Pruning Algorithm

2.3 Genetic Algorithm

Genetic Algorithms (GAs) are inspired by the process of natural selection and are used to find approximate solutions to optimization problems. They work by evolving a population of candidate solutions through selection, crossover, and mutation operations. In the context of the game, GAs can be used to evolve strategies or decision-making heuristics by iteratively improving the population of solutions based on a fitness function.

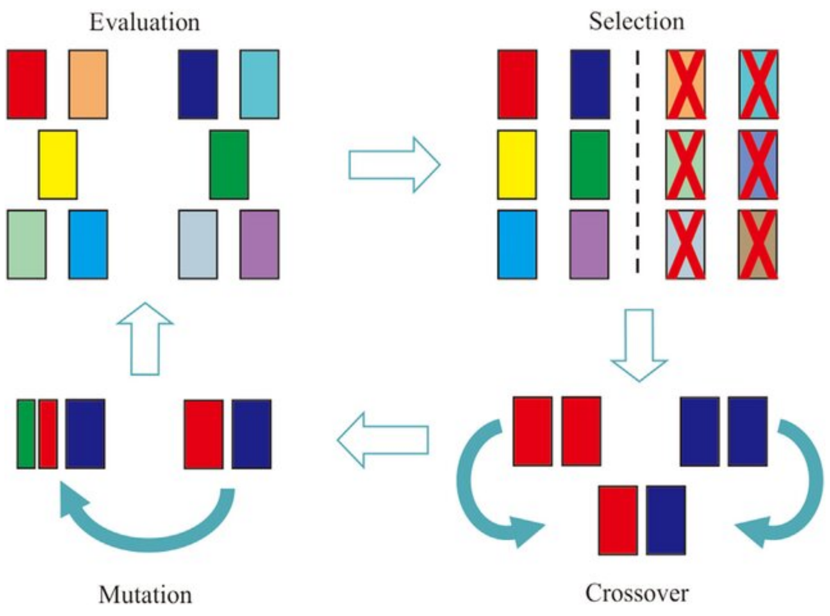


Figure 2.3: Genetic Algorithm

2.4 Fuzzy Logic

Fuzzy Logic is a form of many-valued logic that deals with reasoning that is approximate rather than fixed and exact. In the game application, Fuzzy Logic can be employed to make decisions based on uncertain or imprecise information. By using fuzzy sets and membership functions, the algorithm can handle the complexity of decision-making in scenarios where input variables are not strictly binary.

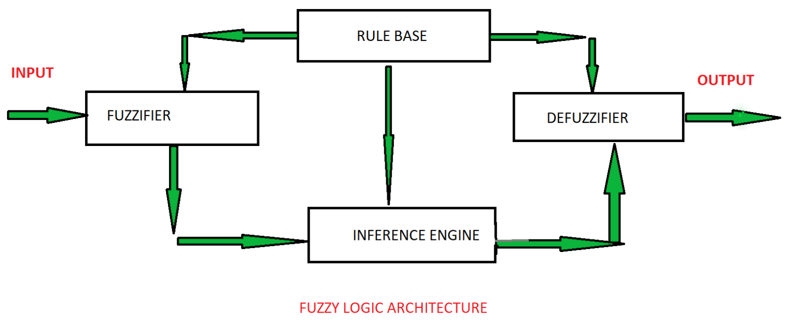


Figure 2.4: Fuzzy Logic Algorithm

2.5 A* Algorithm

The A* (A-star) algorithm is a pathfinding and graph traversal algorithm that is widely used in game development for finding the shortest path from a start node to a goal node. It combines features of Dijkstra's Algorithm and Greedy Best-First Search by using a heuristic to estimate the cost to reach the goal, thereby improving the efficiency of the search process. The A* algorithm is particularly useful in scenarios where the game requires navigation or pathfinding.

Feel free to adjust or expand on any of these sections based on the specifics of your implementation and the depth of detail you want to include.

CHAPTER III

Methodology

3.1 Game Rules and Setup

The DoRa game is played on an $n \times n$ matrix board. Each player is assigned a move type after the initial toss:

- **Vertical Move:** Covers two consecutive cells vertically.
- **Horizontal Move:** Covers two consecutive cells horizontally.

Players alternate turns to place their respective moves on the board, aiming to fill the board while minimizing the opponent's possible moves.

3.2 MinMax Algorithm

The MinMax algorithm is employed to determine the optimal move by recursively evaluating the game tree. The algorithm assumes that the player is trying to maximize their advantage while the opponent tries to minimize it. The objective is to choose a move that maximizes the player's minimum gain, thereby minimizing the opponent's potential moves.
To apply the MinMax algorithm, the game state is recursively explored, and the algorithm evaluates the potential future game states to determine the move that leads to the most favorable outcome for the player.

3.3 Alpha-Beta Pruning

Alpha-Beta Pruning is an optimization of the MinMax algorithm designed to reduce the number of nodes evaluated in the game tree. It uses two parameters, Alpha and Beta, to cut off branches that cannot affect the final decision:

- **Alpha:** The best value that the maximizing player can guarantee so far.

- **Beta:** The best value that the minimizing player can guarantee so far.

During the search, branches are pruned if their values cannot improve the current Alpha or Beta values, thereby improving the efficiency of the search process.

3.4 Genetic Algorithm

The Genetic Algorithm (GA) is used to optimize the move selection process by simulating evolution. The algorithm involves the following steps:

- **Initialization:** Generate an initial population of potential moves.
- **Evaluation:** Assess the fitness of each move based on its ability to minimize the opponent's legal moves.
- **Selection:** Choose the best-performing moves to act as parents for the next generation.
- **Crossover and Mutation:** Combine and alter parent moves to create new move combinations.

The Genetic Algorithm iterates through generations of moves to evolve strategies that effectively minimize the opponent's legal moves.

3.5 Fuzzy Logic

Fuzzy Logic is utilized to handle uncertainty and approximate reasoning in decision-making. In the context of the game:

- Moves are evaluated based on their position on the board, with higher scores assigned to moves closer to the center and lower scores to moves near the edges.

The Fuzzy Logic approach helps to make more nuanced decisions by incorporating imprecise information and preferences into the move evaluation process.

3.6 A* Algorithm

The A* Algorithm is used for pathfinding and decision-making:

- **Search Process:** Uses a priority queue to explore the game states based on their cost and heuristic estimates.
- **Heuristic Function:** Estimates the number of remaining moves to guide the search towards the goal efficiently.

The A* Algorithm aims to find the most efficient path to a desirable game state by considering both the cost of moves and heuristic estimates.

CHAPTER IV

Design

4.1 Main/Menu Page

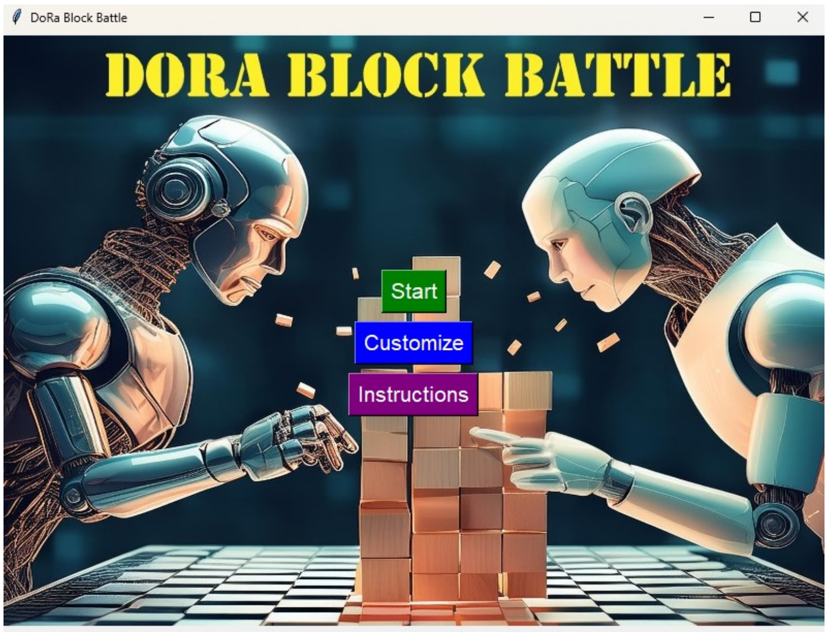


Figure 4.1: Main Page

The main/menu page serves as the entry point for the application and provides the following options to the user:

- **Start:** Clicking the "Start" button redirects the user to the toss page.
- **Customize:** Allows the user to customize game settings and preferences.
- **Instructions:** Provides information and guidance on how to play the game.

The main/menu page is designed to offer easy navigation to the core functionalities of the application, ensuring a smooth user experience from the start.

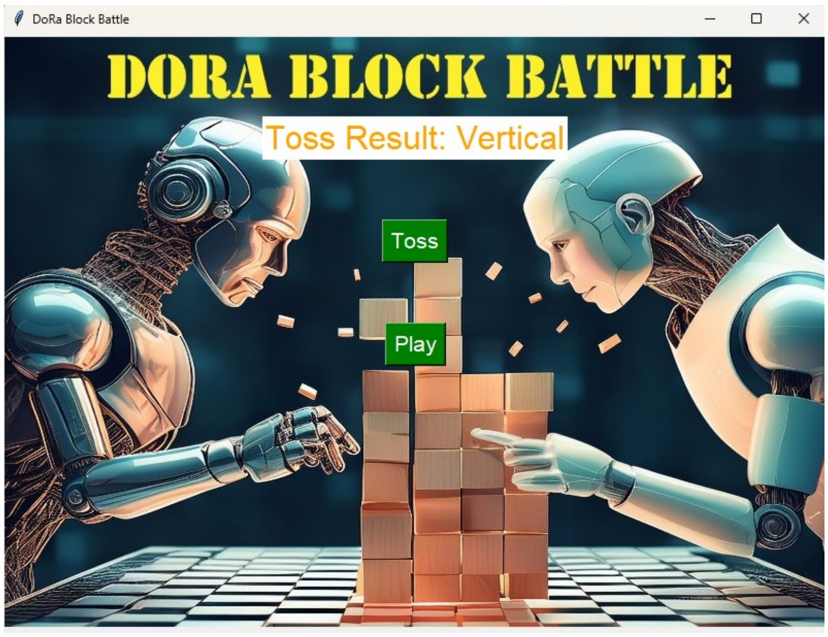


Figure 4.2: Toss Page

4.2 Toss Page

After clicking the "Start" button on the main/menu page, the user is directed to the toss page. Here, the user performs a toss to determine the move type (Vertical or Horizontal). Based on the result of the toss:

- The user is assigned a specific move type (Vertical or Horizontal).
- The opponent is assigned the remaining move type.

The toss page ensures that the game begins with a fair assignment of move types to both the user and the opponent.

4.3 Game Board Page

The game board page displays the main game interface where the gameplay takes place. It includes:

- **View 1:** This view shows the initial state of the game board, including the layout and any static elements.
- **View 2:** This view illustrates the game board with additional details or in a different state, providing further insight into the game's functionality.

These views offer a comprehensive look at the game board, showcasing the design and layout elements critical to the gameplay experience.

4.4 Customize Page

The customize page allows users to adjust the game settings to their preferences. Options available on this page include:

- **Board Size:** Adjust the dimensions of the game board.
- **Move Types:** Customize the types of moves available.
- **Other Settings:** Modify additional game settings as desired.

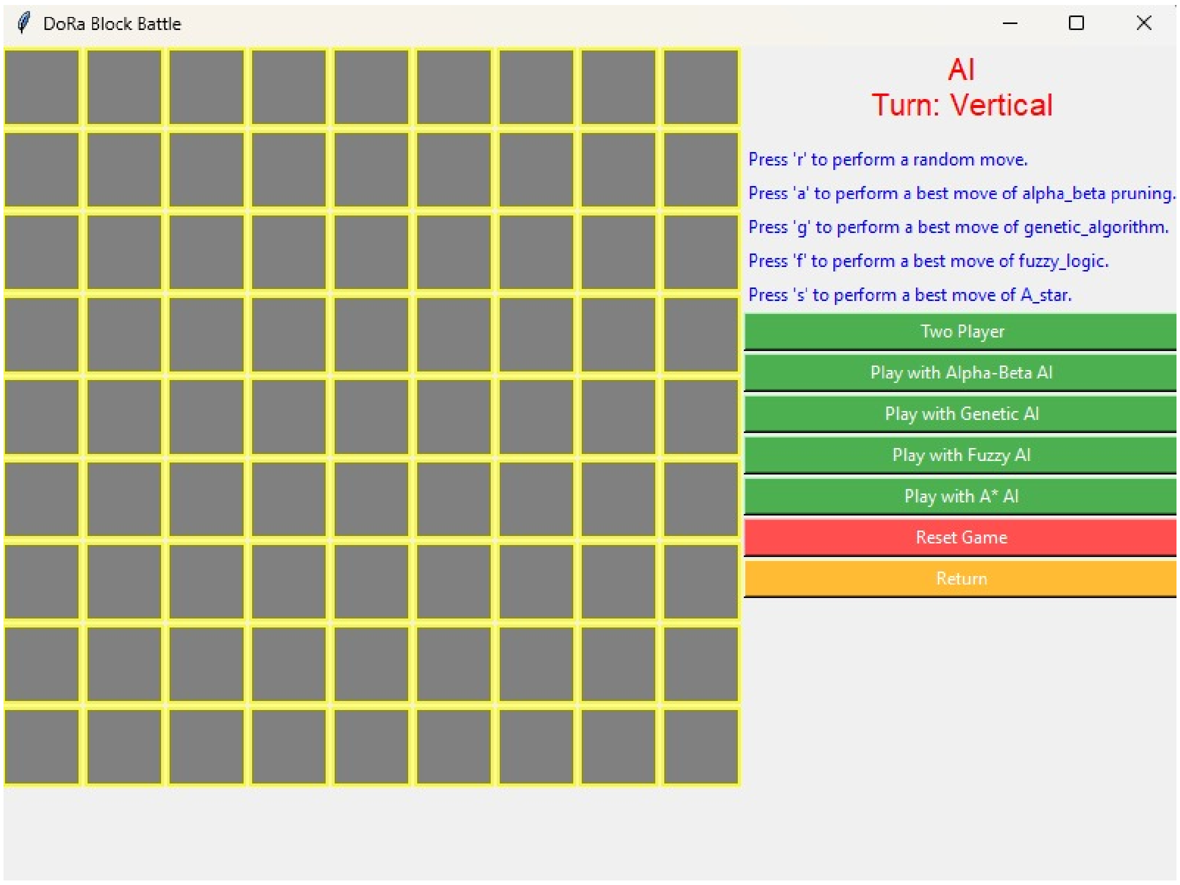
This page provides a user-friendly interface for modifying game configurations to enhance the gaming experience.

4.5 Instructions Page

The instructions page provides essential guidance and rules for playing the game. It includes:

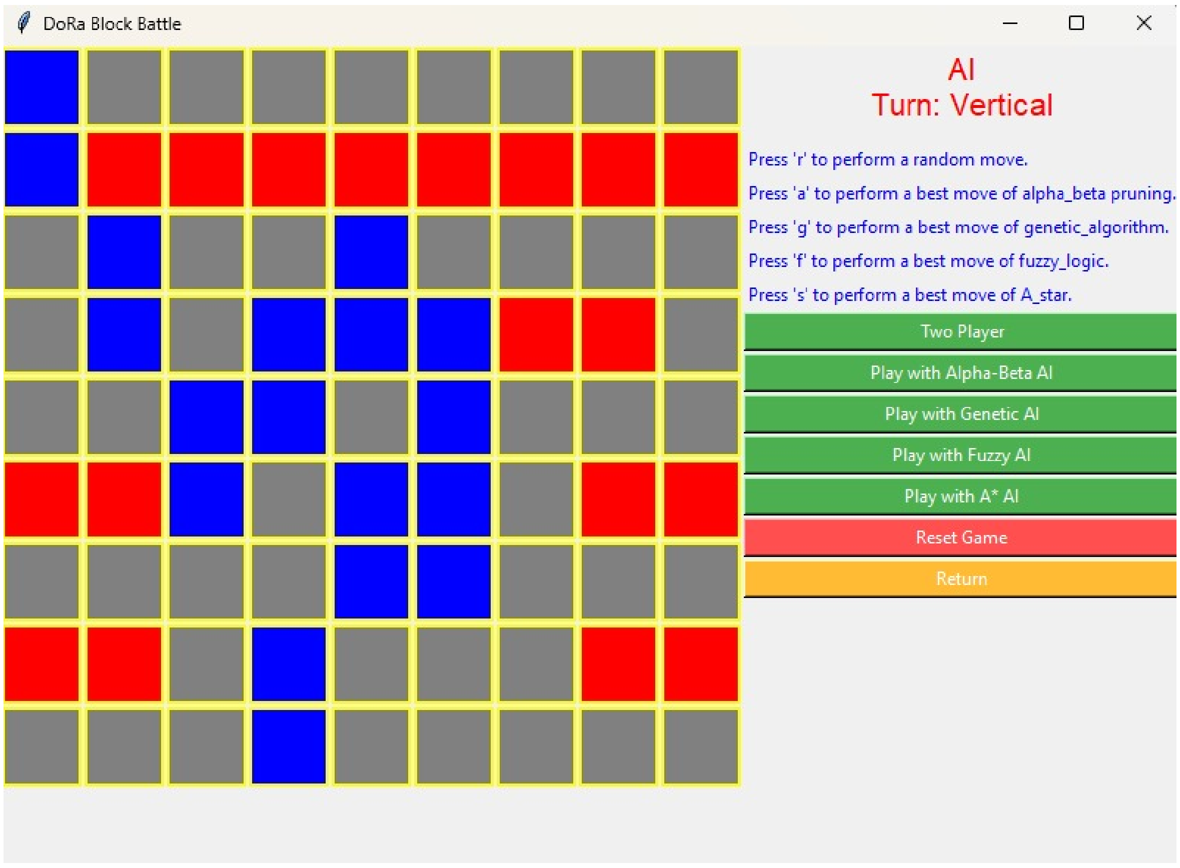
- **Gameplay Instructions:** Step-by-step guide on how to play the game.
- **Control Commands:** Key commands and their functions.
- **Additional Tips:** Helpful hints for improving gameplay.

This page ensures that users have access to all necessary information to play the game effectively.



Press 'r' to perform a random move.
Press 'a' to perform a best move of alpha_beta pruning.
Press 'g' to perform a best move of genetic_algorithm.
Press 'f' to perform a best move of fuzzy_logic.
Press 's' to perform a best move of A_star.

Figure 4.3: Game Board Page - View 1



Press 'r' to perform a random move.
Press 'a' to perform a best move of alpha_beta pruning.
Press 'g' to perform a best move of genetic_algorithm.
Press 'f' to perform a best move of fuzzy_logic.
Press 's' to perform a best move of A_star.

Figure 4.4: Game Board Page - View 2

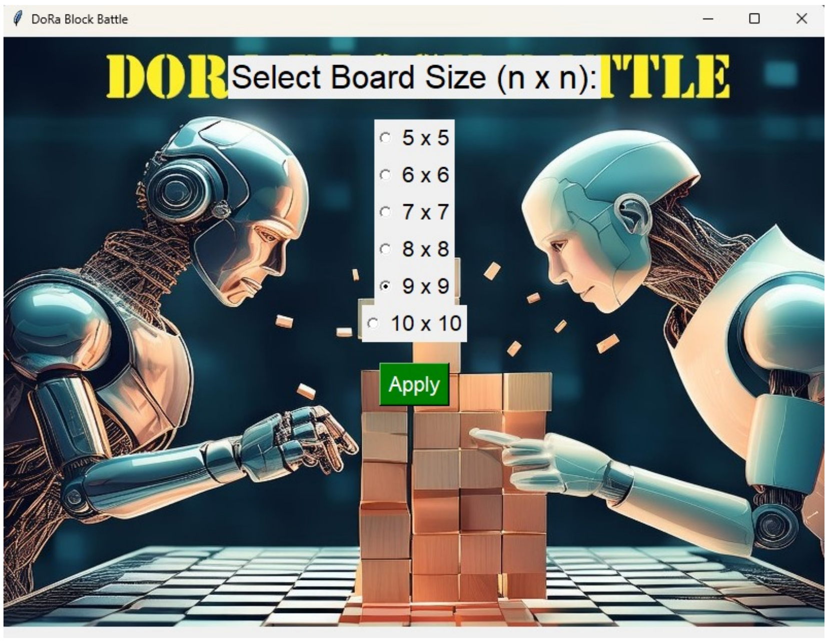
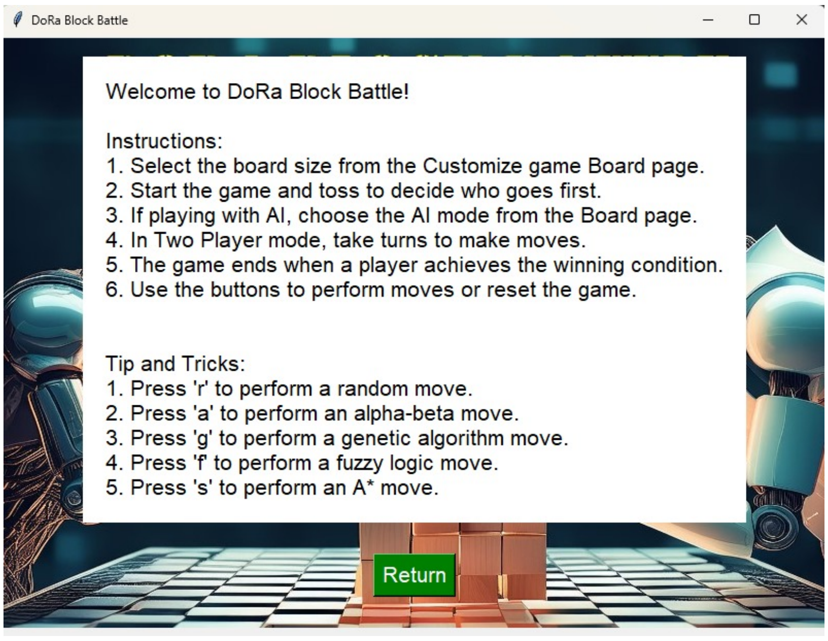


Figure 4.5: Customize Page



Welcome to DoRa Block Battle!

Instructions:
1. Select the board size from the Customize game Board page.
2. Start the game and toss to decide who goes first.
3. If playing with AI, choose the AI mode from the Board page.
4. In Two Player mode, take turns to make moves.
5. The game ends when a player achieves the winning condition.
6. Use the buttons to perform moves or reset the game.

Tip and Tricks:
1. Press 'r' to perform a random move.
2. Press 'a' to perform an alpha-beta move.
3. Press 'g' to perform a genetic algorithm move.
4. Press 'f' to perform a fuzzy logic move.
5. Press 's' to perform an A* move.

Figure 4.6: Instructions Page

CHAPTER V

Implementation

5.1 MinMax Algorithm

The MinMax algorithm is implemented to determine the optimal move by evaluating potential future game states. It recursively explores the game tree and evaluates the potential outcomes to select the best move.

Pseudocode:

```
function MinMax(node, depth, isMaximizingPlayer):
    if depth == 0 or node is a terminal node:
        return the heuristic value of node

    if isMaximizingPlayer:
        bestValue = -
        for each child of node:
            value = MinMax(child, depth - 1, false)
            bestValue = max(bestValue, value)
        return bestValue
    else:
        bestValue =
        for each child of node:
            value = MinMax(child, depth - 1, true)
            bestValue = min(bestValue, value)
        return bestValue
```

5.2 Alpha-Beta Pruning

Alpha-Beta Pruning is used to optimize the MinMax algorithm by reducing the number of nodes evaluated in the game tree. It uses two parameters, Alpha and Beta, to cut off branches that cannot influence the final decision.

Pseudocode:

```
function AlphaBeta(node, depth, alpha, beta, isMaximizingPlayer):
    if depth == 0 or node is a terminal node:
        return the heuristic value of node

    if isMaximizingPlayer:
        bestValue = -
        for each child of node:
            value = AlphaBeta(child, depth - 1, alpha, beta, false)
            bestValue = max(bestValue, value)
            alpha = max(alpha, bestValue)
            if beta <= alpha:
                break
        return bestValue
    else:
        bestValue =
        for each child of node:
            value = AlphaBeta(child, depth - 1, alpha, beta, true)
            bestValue = min(bestValue, value)
            beta = min(beta, bestValue)
            if beta <= alpha:
                break
        return bestValue
```

5.3 Genetic Algorithm

The Genetic Algorithm is utilized to optimize move selection by simulating evolution through a population of potential moves.

Pseudocode:

```
function GeneticAlgorithm():
    population = InitializePopulation()
    for generation in range(maxGenerations):
        fitnessScores = EvaluateFitness(population)
        parents = SelectParents(population, fitnessScores)
        offspring = Crossover(parents)
        offspring = Mutate(offspring)
        population = ReplaceOldPopulation(population, offspring)
    return BestSolution(population)

function InitializePopulation():
    # Generate an initial population of random moves
    return population

function EvaluateFitness(population):
    # Assess the fitness of each move in the population
    return fitnessScores

function SelectParents(population, fitnessScores):
    # Select the best-performing moves to act as parents
    return parents

function Crossover(parents):
    # Combine parent moves to create new move combinations
    return offspring

function Mutate(offspring):
    # Apply mutations to introduce diversity
    return mutatedOffspring

function ReplaceOldPopulation(population, offspring):
    # Replace old population with new offspring
    return newPopulation
```

```
function BestSolution(population):
    # Find and return the best solution from the population
    return bestMove
```

5.4 Fuzzy Logic

Fuzzy Logic is employed to handle imprecision in decision-making, evaluating moves based on their position and assigning scores.

Pseudocode:

```
function FuzzyLogicEvaluate(move):
    score = 0
    if move is near center of board:
        score += highValue
    if move is near edges:
        score -= lowValue
    return score

function FuzzyLogicDecision(moves):
    scores = []
    for move in moves:
        score = FuzzyLogicEvaluate(move)
        scores.append(score)
    return MoveWithHighestScore(scores)
```

5.5 A* Algorithm

The A* Algorithm is used for pathfinding and decision-making, considering both cost and heuristic estimates.

Pseudocode:

```
function AStar(start, goal):
    openSet = PriorityQueue()
    openSet.add(start, 0)
    cameFrom = {}
    gScore = InitializeScores()
    gScore[start] = 0
    fScore = InitializeScores()
    fScore[start] = Heuristic(start, goal)

    while not openSet.isEmpty():
        current = openSet.pop()
        if current == goal:
            return ReconstructPath(cameFrom, current)

        for neighbor in GetNeighbors(current):
            tentativeGScore = gScore[current] + Cost(current, neighbor)
            if tentativeGScore < gScore[neighbor]:
                cameFrom[neighbor] = current
                gScore[neighbor] = tentativeGScore
                fScore[neighbor] = gScore[neighbor] + Heuristic(neighbor, goal)
                if not openSet.contains(neighbor):
                    openSet.add(neighbor, fScore[neighbor])

    return failure

function ReconstructPath(cameFrom, current):
    path = [current]
    while current in cameFrom:
        current = cameFrom[current]
        path.append(current)
    return path[::-1]

function Heuristic(node, goal):
    # Estimate the cost from node to goal
    return heuristicCost
```

CHAPTER V

Results and Discussion

5.1 Performance Comparison

In this section, we evaluate and compare the performance of various AI algorithms used in the game application. The comparison is based on the effectiveness and efficiency of each algorithm in making optimal decisions. The algorithms evaluated are Alpha Beta Pruning, Genetic Algorithm, Fuzzy Logic, and A*.

Algorithm	**Performance**
Alpha Beta Pruning	**Best**
Genetic Algorithm	**Good**
Fuzzy Logic	**Moderate**
A*	**Least Effective**

Table 5.1: Comparison of AI Algorithms Based on Performance

The table above summarizes the relative performance of each AI algorithm in our game application. Based on our experiments and evaluations, the following conclusions can be drawn:

- **Alpha Beta Pruning:** This algorithm consistently provided the best performance due to its comprehensive evaluation of all possible moves, leading to optimal decisions in various game scenarios.
- **Genetic Algorithm:** The Genetic Algorithm demonstrated good performance by evolving effective strategies over generations. It provided a balance between exploration and exploitation of moves.
- **Fuzzy Logic:** While Fuzzy Logic provided a reasonable level of decision-making capability, it was less effective compared to Alpha Beta Pruning and Genetic Algorithm due to its reliance on heuristic rules and approximate reasoning.

- **A* Algorithm:** The A* Algorithm, although effective in pathfinding, was the least effective in the context of this game due to its heuristic-based approach, which did not always align well with the game's strategic needs.

In conclusion, the Alpha Beta Pruning algorithm proved to be the most effective for this game application, followed by the Genetic Algorithm, Fuzzy Logic, and A* Algorithm. These findings highlight the importance of choosing the right AI approach based on the specific requirements and complexity of the game.

CHAPTER V

Conclusion

5.1 Summary of Findings

This project aimed to develop a game application using Tkinter and various artificial intelligence (AI) algorithms, including MinMax, Genetic Algorithm, Fuzzy Logic, and A*. The game mechanics were designed to ensure fairness and strategic depth, with the primary focus on optimizing the decision-making process for both the player and the AI.
The implementation of the AI algorithms provided valuable insights into their effectiveness:

- **MinMax Algorithm:** Demonstrated superior performance in determining optimal moves by evaluating the entire game tree. It consistently delivered the best results, making it the most reliable algorithm for this project.

- **Genetic Algorithm:** Showed strong performance in optimizing move selection through evolutionary processes. Although not as effective as MinMax, it provided a good balance between computational efficiency and move quality.

- **Fuzzy Logic:** Offered a more nuanced decision-making process by incorporating imprecise information. While less accurate than MinMax and Genetic Algorithm, it contributed to more flexible and adaptive AI behavior.

- **A* Algorithm:** Although useful for pathfinding and heuristic-based searches, it was less effective in this context compared to the other algorithms. Its performance was satisfactory but did not match the efficiency of MinMax or Genetic Algorithm.

5.2 Conclusions

In summary, the MinMax algorithm proved to be the most effective AI strategy for this game, providing optimal move decisions by thoroughly exploring possible outcomes. The Genetic Algorithm, while not as powerful as MinMax, still performed well and offered a practical approach to move optimization. Fuzzy Logic added a layer of adaptability, though it was less precise compared to MinMax and Genetic Algorithm. The A* Algorithm, while beneficial in certain scenarios, did not offer significant advantages in the context of this project.

The results highlight the importance of selecting the appropriate AI algorithm based on the specific requirements and constraints of the game. Future work could explore hybrid approaches that combine the strengths of these algorithms to further enhance game performance and AI capabilities.

Overall, this project has demonstrated the effectiveness of different AI techniques in a practical application and has laid the groundwork for future research and development in AI-driven game design.

References

References

1. GeeksforGeeks. (n.d.). *Minimax Algorithm in Game Theory - Set 1 (Introduction)*. Retrieved from https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

2. JavaTpoint. (n.d.). *AI - Alpha Beta Pruning*. Retrieved from https://www.javatpoint.com/ai-alpha-beta-pruning

3. GeeksforGeeks. (n.d.). *Genetic Algorithms*. Retrieved from https://www.geeksforgeeks.org/genetic-algorithms/

4. GeeksforGeeks. (n.d.). *Fuzzy Logic - Introduction*. Retrieved from https://www.geeksforgeeks.org/fuzzy-logic-introduction/

5. Codecademy. (n.d.). *A* Search Algorithm*. Retrieved from https://www.codecademy.com/resources/docs/ai/search-algorithms/a-star-search