**Project Report: Nine Man Morris**

By

**Ehsanul Karim**

Roll: 1907039

&

**Afiat Khan Tahsin**

Roll: 1907047



**Submitted To:**

Md. Shahidul Salim

Lecturer

Signature

Most. Kaniz Fatema Isha

Lecturer

Signature

**Department of Computer Science and Engineering**

**Khulna University of Engineering & Technology**

**Khulna 9203, Bangladesh**

**3rd September 2024**

1. Introduction
   1. Background

The Nine Men's Morris is an ancient strategy board game that’s been played for centuries. The game involves two players strategically placing and moving pieces to form "mills" (three pieces in a row). Despite having simple rules, a deep strategic thinking is required to win this game. Motivated by the success of AI in board games, we develop this project where we’re developing a digital version of Nine Men's Morris using Python. The AI is implemented using algorithms like Minimax, Alpha-Beta Pruning, and Genetic Algorithms. By analyzing possible future outcomes with these techniques and thus decides the best moves.

* 1. Objectives

The projects major objective is to develop an AI-powered game with implementing advanced algorithms to make the AI capable of strategic gameplay. So, Nine Men's Morris game is implemented as it offers a challenging experience for players. The primary goal of this project is

* Developing the Nine Men's Morris game using the Python Pygame library with integration of advanced AI algorithms.
* Implementing the Minimax algorithm for optimal decision-making in the game.
* Enhancing the AI’s efficiency by incorporating Alpha-Beta Pruning to reduce unnecessary computations.
* Using Genetic Algorithms for evolving strategies.
* Using Fuzzy-Logic to determine the intelligence level of AI
* Designing a user-friendly interface to easily interact.
* Providing a competitive AI opponent
* Gain a deep understanding of game theory principles and how they can be applied to AI based game.
  1. Project planning

The project planning consists of requirement analysis, study of techniques used, improvement of UI/UX, implementation of the two-player game, implementation of AI, Integrating the whole system, and other sections. The relevant details are shown in the following Figure 1.1. The figure contains a Gantt chart, which has details of the work flow with time duration to specifically explain the planning and flow of operations.

**Figure 1.1**: Gantt Chart of the project.

1. Project Design

The Project was designed with the fact that it can be scalable and adaptable, making it flexible for any future addition and modification. The reason is that with newer technology and algorithms, any AI can become increasingly intelligent over time.

* 1. Analysis of the system

The system is analyzed through a flow diagrams to exhibit the sequence of operations.

* + 1. Flow Diagram

The project’s operation consists of multiple steps. The flow diagram, as shown in Figure 3.1, provides a visual breakdown of the various stages in the game's decision-making process, which helps us to understand the system's inner workings and the interactions between different components.

Move Piece

Place Piece

Check Phase

Start

White Player Turn

Black Player Turn

Kill opponent Player

Check Mil

Change Turn

Check Turn

First Second

Yes

No

**Figure 2.1:** DFD diagram of the project.

* 1. System architecture

The construction of the project requires in depth knowledge of its architecture. The structure of the project is exhibited in a class diagram, illustrating the encapsulation and association of key classes.

* + 1. Class Diagram

The information and functionality of association and encapsulation of each individual class or building block is shown in the Figure 3.3 where the blocks resonate the required information.

Game State

+ Board: String<Array>

+ Current Player: String

+ Opponent Player: String

Apply\_move(player,move)

Get\_legal\_moves(player)

Make\_kill(opponent)

Get\_neighbour(position)

Get\_killing\_moves()

Check\_mil()

Chromosome

+ Moves: (int,int)<Array>

Fitness()

<Interface>

Display\_text()

Update\_info()

Update\_time()

Draw\_pieces()

Draw\_rectangles()

Fuzzy Intelligence

+ Time: Date Time

+ Total no moves: Int

+ Black Count: Int

+ White Count: Int

- Score: Int

Seconds\_membership(time)

Number\_move\_membership(total moves)

Guti\_membership(player)

Rule\_evaluation()

Defuzzification()

Genetic Algorithm

- No of Generation: Int

- Population Size: Int

- Gene Size: Int

- Mutation Rate: Float

+ State: Game State

- Chromosomes: Chromosome <Array>

Game\_over (State)

Evaluate\_moves\_score (moves)

Most\_frequent\_tuple (chromosome)

Selection (Chromosome<Array>)

Crossover (chrom1, chrom2)

Mutation (chrom)

Min Max Alpha Beta

- Depth: Int

- Maximizing\_player: Bool

- Alpha: Float

- Beta: Float

- Mill\_found: Bool

- Eval, Max\_eval, Min\_eval: Int

closedMorris(parent\_board,board)

differceInNumberOfMills(board)

differenceInClosedPeaces(board)

differceInPieces(board)

differenceIn2PeacesConfig(board)

differenceIn3PeacesConfig(board)

**Figure 2.2:** Class diagram of the project.

* 1. Tools used

A wide variety of tools and libraries were used to complete the project. Each tool played a significant role.

* + 1. Libraries

The latest python libraries, including Numpy, Pygame, Math, random, datetime, were masterfully leveraged to create a highly organized and efficient structure for the project.

* + 1. VS Code

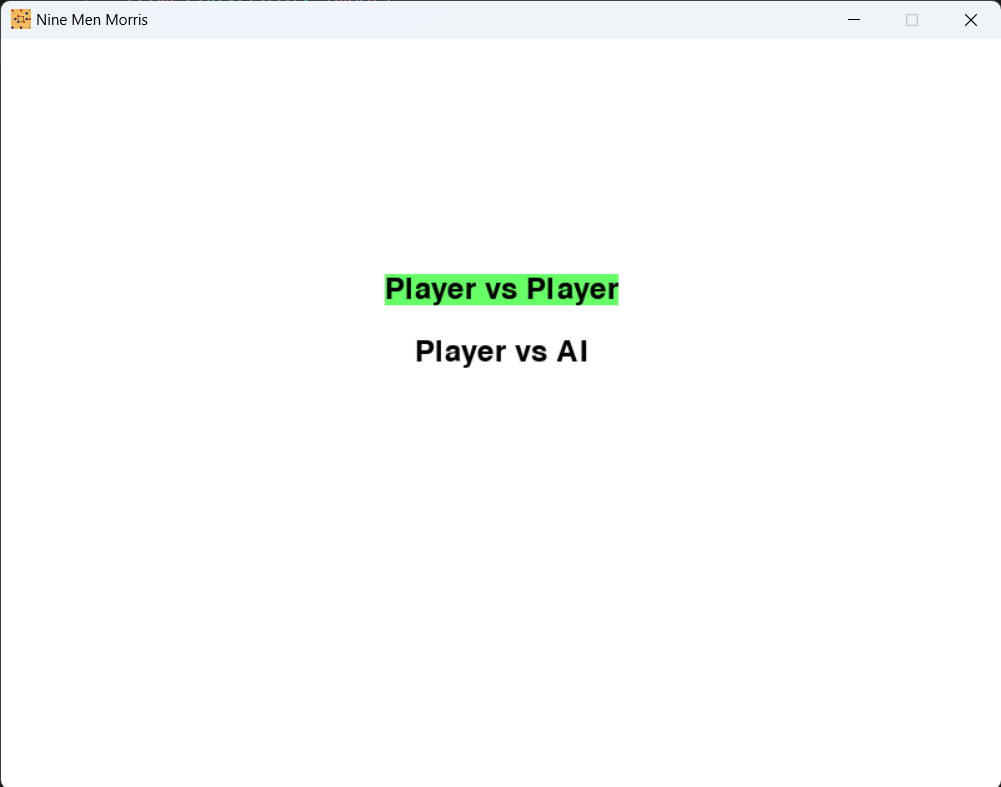
Visual Studio Code serves as a versatile code editor. Various extensions can be found to support both python and user interface which increases coding productivity.

1. Project Implementation

This chapter will describe the steps and individual procedures that was taken to complete the AI project. The game consists of several hard and soft constraints, intelligence algorithms and user-friendly interface which is described in the following.

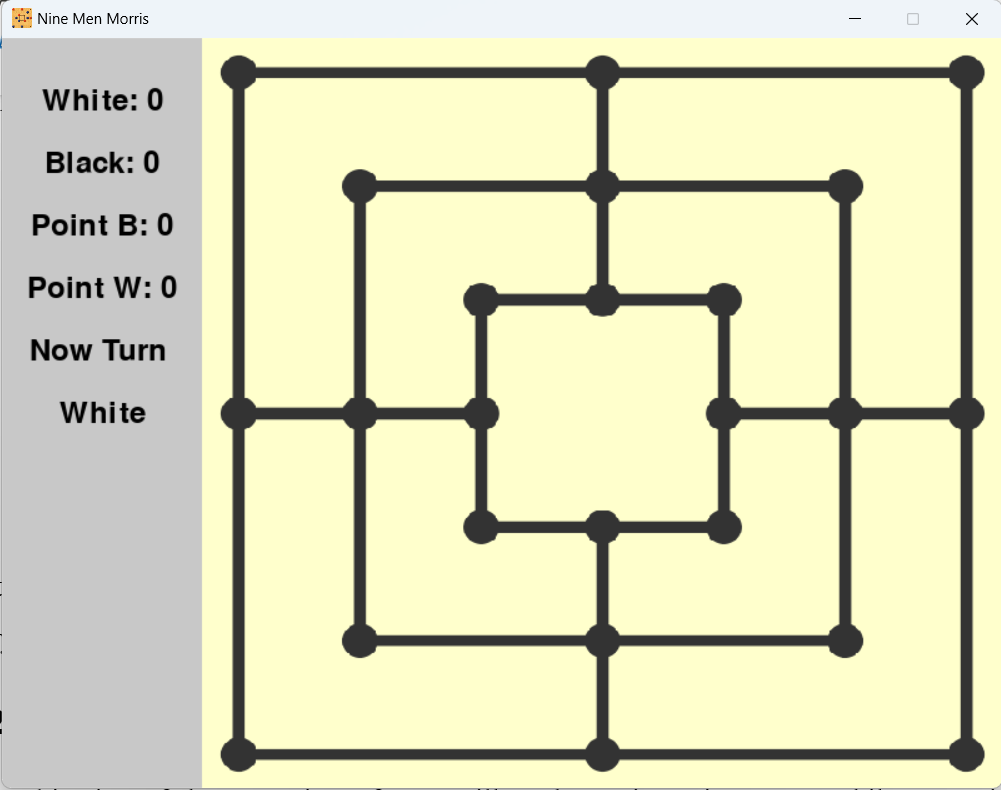
* 1. Game UI

There are two options for a player to choose at first, he can choose to play as player vs player or player (White) vs AI.



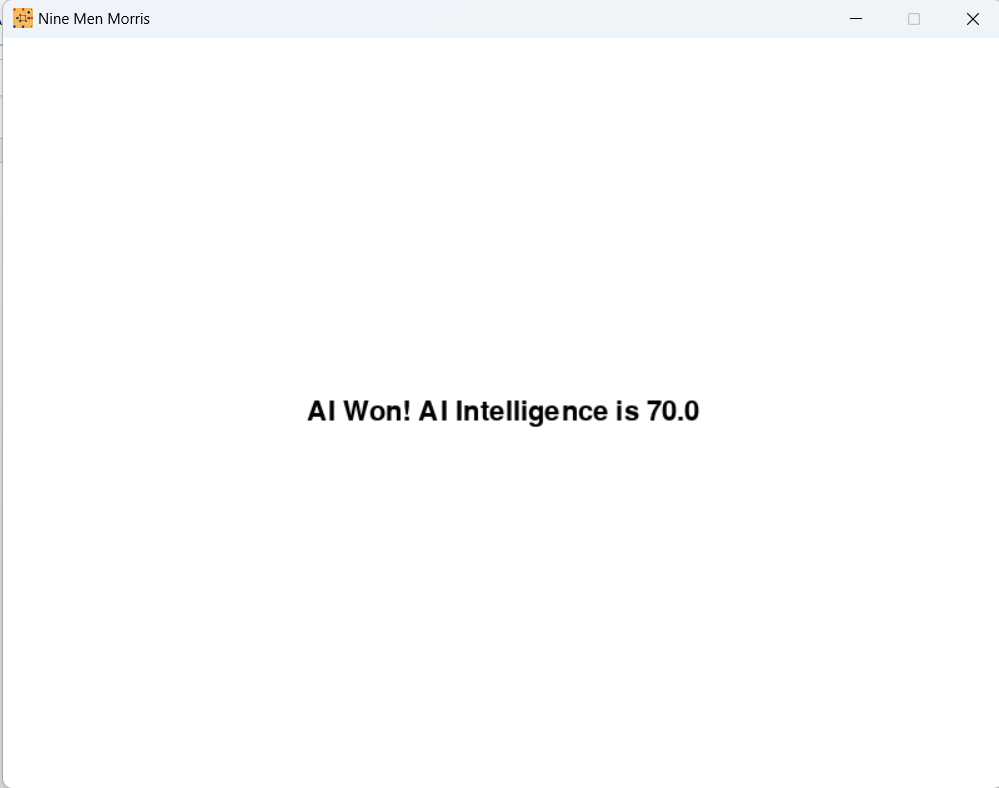
**Figure 3.1:** Initial Game Screen.

The game contains a board consists of three concentric squares connected by lines, forming 24 points where pieces can be placed.



**Figure 3.2:** The Board of Nine Man Morris Game.

After the game finishes, a pop-up box which contains the game verdict and point of the AI player using fuzzy inference will be showed.



**Figure 3.3:** User Interface of Game-verdict.

* 1. Game Rules

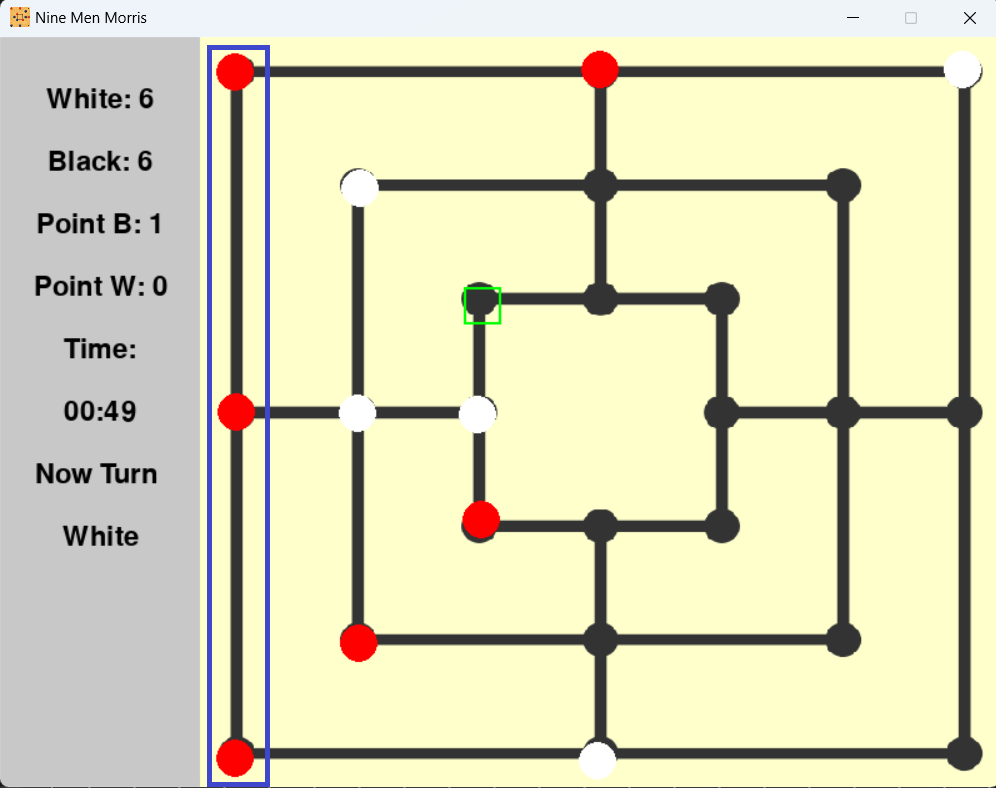
The objective of the game is to form "mills" (three pieces in a row) while strategically blocking the opponent and eventually reducing their number of pieces to two. The game is divided into two main phases: the Setup Phase and the Regular Phase. The details of these procedures are as following.

* + 1. Setup Phase

During this phase, each player takes turns placing one of their nine pieces on any empty point on the board. The goal in this phase is to position player pieces to either form mills or block opponent from doing so. When any player successfully aligns three of their pieces in a straight line they have formed a mill leading them to kill one of the opponent's pieces from the board. Obviously, a piece that is part of an opponent's mill should not be killed unless no other pieces are available to do so.

Once all pieces have been placed on the board, the Setup Phase concludes, and the game transitions into the Regular Phase.

(i) (ii)



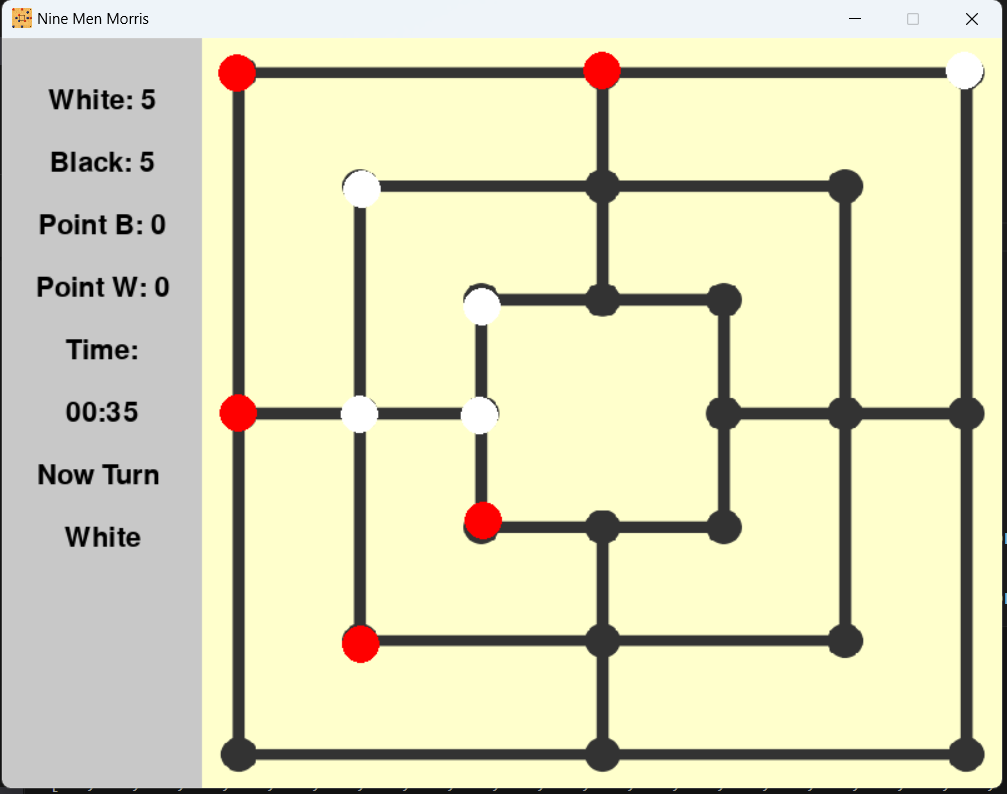


Figure 3.4: Setup Phase. Whenever a mil is made. An opponent player is killed. A Green box showing which piece had been killed.

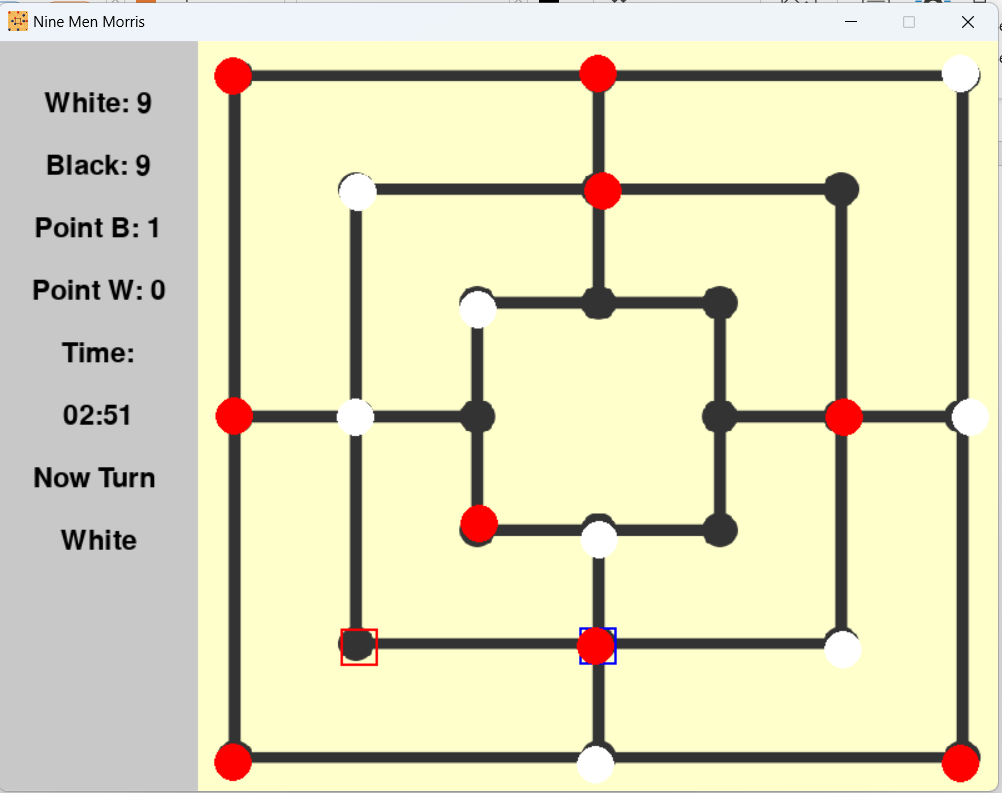
* + 1. Regular Phase

During this phase, each player continues to take turns by moving his pieces to any empty neighboring cells. Similarly, a mill allows the player to remove one of the opponent's pieces.

The game concludes when one player is reduced to only two pieces, as they can no longer form a mill. The opponent is declared the winner. If a player is unable to make a legal move the game is declared as draw.

(ii)

(i)



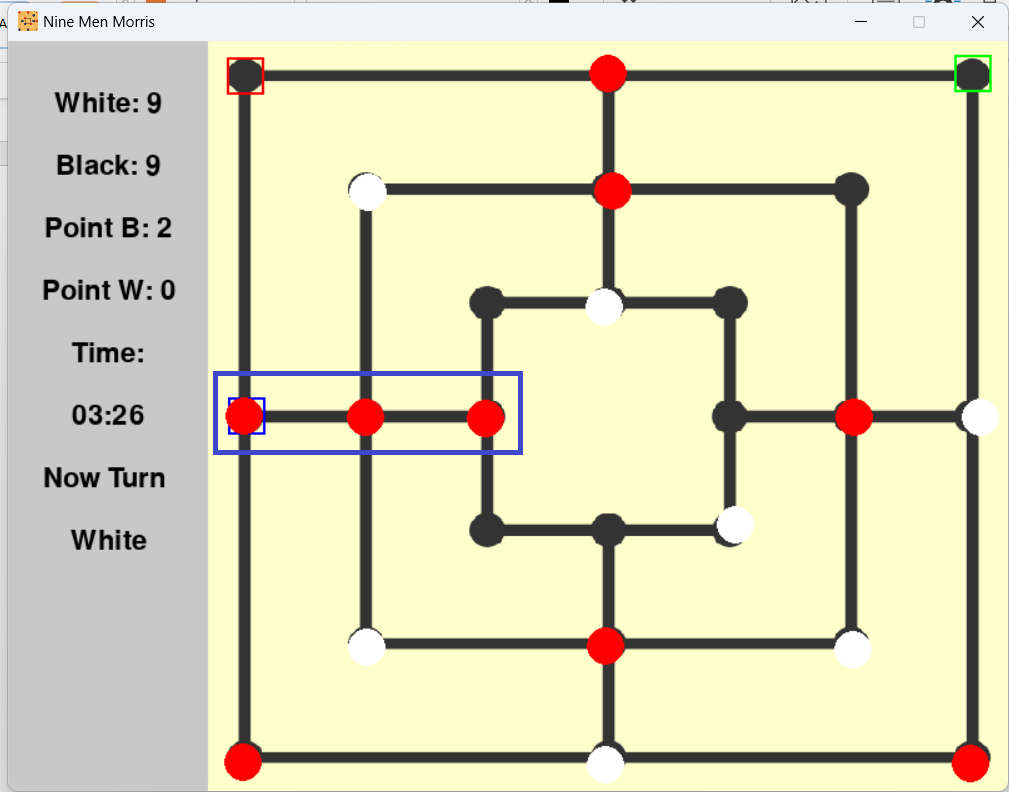


Figure 3.5: Regular Phase. The red rectangle shows where was he moved from and the blue rectangle shows where he placed now.

* 1. Optimizing AI Strategies

This section describes how AI decides his best move and evaluate his performance using different heuristic algorithms, implementation details along with their corresponding pseudo code.

* + 1. The Role of Alpha-Beta Pruning with Minimax

During the first phase, AI optimizes its decision making process by recursively simulates all possible move till depth 5, evaluating each move and selecting the best one based on a heuristic evaluation function. Alpha-Beta pruning enhances the efficiency of this algorithm by pruning branches that cannot possibly influence the final decision.

Pseudo Code:

The algorithm can be described by the following functions.

function MinimaxAlphaBeta (board, depth, maximizingPlayer, alpha, beta, player, phase, parentBoard, millFound, redPlaced, whitePlaced):

Base case: If at maximum depth, game is over, or first phase ends

if depth is 0 or game\_over(board) or first\_phase\_end(redPlaced, whitePlaced):

Evaluate and return board score

// If a mill has been found, handle capturing moves

if millFound:

if maximizingPlayer:

maxEval = -∞

for each possible kill move:

save current board as parentBoard

newBoard = make\_kill(board, selected move)

Reset millFound after the kill

eval = Call (MinimaxAlphaBeta) with depth--

maxEval = max (maxEval, eval)

alpha = max (alpha, eval)

if beta <= alpha:

break // Beta cutoff

return maxEval

else:

minEval = ∞

for each possible kill move:

save current board as parentBoard

newBoard = make\_kill(board, selected move)

Reset millFound after the kill

eval = Call (MinimaxAlphaBeta) with depth--

minEval = min (minEval, eval)

beta = min (beta, eval)

if beta <= alpha:

break // Alpha cutoff

return minEval

// If no mill is found, proceed with normal moves

else:

if maximizingPlayer:

maxEval = -∞

for each possible move:

save current board as parentBoard

newBoard = make\_move(board, selected move, player)

Increment red pieces placed

if check\_mill(newBoard, selected move, player):

millFound = True

eval = Call (MinimaxAlphaBeta) with same depth

else:

millFound = False

eval = Call (MinimaxAlphaBeta) with depth--

maxEval = max (maxEval, eval)

alpha = max (alpha, eval)

if beta <= alpha:

break // Beta cutoff

return maxEval

else:

minEval = ∞

for each possible move:

save current board as parentBoard

newBoard = make\_move(board, selected move, opponent(player))

// Increment white pieces placed

if check\_mill(newBoard, selected move, opponent(player)):

millFound = True

eval = Call (MinimaxAlphaBeta) with same depth

else:

millFound = False

eval = Call (MinimaxAlphaBeta) with same depth--

minEval = min(minEval, eval)

beta = min(beta, eval)

if beta <= alpha:

break // Alpha cutoff

return

* + 1. The Role of Genetic Algorithm

In the second phase of the game, the AI optimizes its decision-making process using a Genetic Algorithm (GA). The GA generates a population of possible moves, which are then evaluated based on a heuristic function. Through processes like selection, crossover, and mutation, the GA iteratively refines these moves, seeking the most effective strategy. This approach allows the AI to explore a wide range of potential moves and evolve better strategies over generations. By focusing on the most promising options, the GA efficiently guides the AI towards the best possible decisions in the game’s complex mid-phase.

Pseudo Code:

The algorithm can be described by the following functions.

def genetic\_algorithm(board, current\_player, depths):

# Initialize parameters

POPULATION\_SIZE = 10

GENE\_SIZE = 10

GENERATIONS = 10

MUTATION\_RATE = 0.1

# Initialize memory to store move evaluations

Memory = {}

# Define helper functions

def crossover(parent1, parent2):

# Create two new children by combining genes from the parents

child1, child2 = combine\_genes(parent1, parent2)

return child1, child2

def mutate(chromosome):

# Randomly alter a gene in the chromosome based on mutation rate

if should\_mutate(MUTATION\_RATE):

change\_random\_gene(chromosome)

def random\_move():

# Generate a random legal move for the current player

return pick\_random\_legal\_move()

def most\_frequent\_tuple(moves):

# Find and return the move that appears most frequently in the list

return find\_most\_common\_move(moves)

def all\_legal\_moves():

# Get all possible legal moves for the current player

return generate\_all\_legal\_moves()

def find\_score\_of\_legal\_moves(moves):

# Evaluate each move and store the score in Memory

for move in moves:

score = evaluate\_move(move)

Memory[move] = score

# Generate all legal moves and evaluate them

legal\_moves = all\_legal\_moves()

find\_score\_of\_legal\_moves(legal\_moves)

# Initialize population with random chromosomes

population = generate\_random\_population(POPULATION\_SIZE)

# Run the genetic algorithm for a set number of generations

for generation in range(GENERATIONS):

new\_population = []

# Calculate fitness for each chromosome

fitness\_scores = [chromosome.fitness() for chromosome in population]

# Generate a new population

while len(new\_population) < POPULATION\_SIZE:

# Select two parents based on fitness

parent1, parent2 = select\_parents(population, fitness\_scores)

# Create children through crossover

child1, child2 = crossover(parent1, parent2)

# Apply mutation to the children

mutate(child1)

mutate(child2)

# Add children to the new population

new\_population.extend([child1, child2])

# Replace old population with the new one

population = new\_population

# Select the best chromosome based on fitness

best\_chromosome = find\_best\_chromosome(population)

# Return the move that appears most frequently in the best chromosome

return most\_frequent\_tuple(best\_chromosome.moves)

* + 1. The Role of Fuzzy Logic

During the second phase, the AI assesses the player's performance using a fuzzy logic-based system. It evaluates the timing of moves, the total number of moves made, and the pieces on the board. This system assigns membership values to these factors, and based on the fuzzy logic rules, it calculates a performance score for the player, determining whether the player's actions were excellent, good, moderate, bad, or worst.

Pseudo Code:

The algorithm can be described by the following functions.

Function seconds\_membership(crisp\_second):

Initialize degree as an empty dictionary with keys: 'very\_early', 'early', 'ok', 'late', 'very\_late' and set all to 0

If 0 <= crisp\_second < 15:

degree['very\_early'] = 1

Else If 15 <= crisp\_second < 20:

degree['very\_early'] = Decrease from 1 as crisp\_second increases

degree['early'] = Increase from 0 as crisp\_second increases

Else If 20 <= crisp\_second < 25:

degree['early'] = Increase to 1 as crisp\_second increases

Else If 25 <= crisp\_second < 27:

degree['early'] = Decrease from 1 as crisp\_second increases

Else If 27 <= crisp\_second < 30:

degree['early'] = Decrease from 1 as crisp\_second increases

degree['ok'] = Increase from 0 as crisp\_second increases

Else If 30 <= crisp\_second < 34:

degree['ok'] = Increase to 1 as crisp\_second increases

Else If 34 <= crisp\_second < 40:

degree['ok'] = Decrease from 1 as crisp\_second increases

degree['late'] = Increase from 0 as crisp\_second increases

Else If 40 <= crisp\_second < 45:

degree['late'] = Increase to 1 as crisp\_second increases

Else If 45 <= crisp\_second < 55:

degree['late'] = 1

Else If 55 <= crisp\_second < 65:

degree['late'] = Decrease from 1 as crisp\_second increases

degree['very\_late'] = Increase from 0 as crisp\_second increases

Else If crisp\_second >= 65:

degree['very\_late'] = 1

Return degree

# Function to get membership values for total number of moves

Function total\_number\_move\_membership(n):

Initialize degree as an empty dictionary with keys: 'low', 'avg', 'huge' and set all to 0

If 0 <= n < 7:

degree['low'] = 1

Else If 7 <= n < 10:

degree['low'] = Decrease from 1 as n increases

degree['avg'] = Increase from 0 as n increases

Else If 10 <= n < 11:

degree['avg'] = 1

Else If 11 <= n < 12:

degree['avg'] = Decrease from 1 as n increases

degree['huge'] = Increase from 0 as n increases

Else If 12 <= n < 13:

degree['huge'] = Increase to 1 as n increases

Else If n >= 13:

degree['huge'] = 1

Return degree

# Function to get membership values for number of pieces on the board

Function guti\_membership(n):

Initialize degree as an empty dictionary with keys: 'poor', 'good', 'best' and set all to 0

If 0 <= n < 3:

degree['poor'] = 1

Else If 3 <= n < 5:

degree['poor'] = Decrease from 1 as n increases

degree['good'] = Increase from 0 as n increases

Else If 5 <= n < 7:

degree['good'] = Decrease from 1 as n increases

degree['best'] = Increase from 0 as n increases

Else If n >= 7:

degree['best'] = 1

Return degree

# Function to evaluate the rating of the black player

Function evaluate\_black\_player\_rating(sec\_degree, M\_degree, W\_degree, B\_degree):

Initialize rating as an empty dictionary with keys: 'worst', 'bad', 'moderate', 'good', 'excellent' and set all to 0

rating['excellent'] = Compute maximum of specific fuzzy logic rules

rating['good'] = Compute maximum of specific fuzzy logic rules

rating['moderate'] = Compute maximum of specific fuzzy logic rules

rating['bad'] = Compute maximum of specific fuzzy logic rules

rating['worst'] = Compute maximum of specific fuzzy logic rules

Return rating

# Function to find the fuzzy score based on time

Function find\_fuzzy\_score(time):

Convert time to seconds since start (crisp\_second)

Get membership degrees for seconds (sec\_degree)

Get membership degrees for total number of moves (M\_degree)

Get membership degrees for white pieces (W\_degree)

Get membership degrees for black pieces (B\_degree)

Compute player ratings using fuzzy logic (ratings\_detail)

Convert ratings to percentage and calculate weighted score (score)

Return score

1. Conclusion

The project successfully developed a sophisticated AI-powered Nine Men's Morris game, leveraging Minimax, Genetic Algorithms, and Fuzzy Logic for strategic gameplay. Despite challenges in implementation and optimization, the integration of these advanced techniques led to a responsive and competitive AI, significantly enhancing the overall gaming experience.

* 1. Conclusion and challenges faced

The Nine Men's Morris game leverages a multi-phase AI strategy to enhance gameplay and decision-making. In the initial phase, the Minimax algorithm provides a robust framework for evaluating potential moves, ensuring optimal decision-making through recursive simulations. The subsequent phase employs Genetic algorithms to refine and evolve strategies, allowing the AI to adapt and optimize its performance over time. Additionally, fuzzy logic is used to assess the player's performance dynamically, offering a nuanced evaluation of gameplay based on timing, move count, and board state.

While the integration of these advanced techniques has significantly improved the game's AI, several challenges were encountered. Implementing the Minimax algorithm required managing a complex game tree and optimizing performance to handle the computational load. The Genetic algorithm, while powerful, posed challenges in tuning parameters and ensuring convergence to effective strategies. Additionally, incorporating fuzzy logic introduced complexity in designing membership functions and evaluation criteria. Despite these obstacles, addressing them head-on led to a more sophisticated and responsive AI system, ultimately enhancing the overall gaming experience.

# References

1. [Game theory — Minimax. This article will be a bit different… | by NerdzLab | Towards Data Science](https://towardsdatascience.com/game-theory-minimax-f84ee6e4ae6e)
2. [Genetic Algorithms - GeeksforGeeks](https://www.geeksforgeeks.org/genetic-algorithms/)
3. [Fuzzy Logic: Definition, Meaning, Examples, and History (investopedia.com)](https://www.investopedia.com/terms/f/fuzzy-logic.asp#:~:text=Fuzzy%20logic%20is%20a%20set,from%20this%20kind%20of%20data.)
4. [The Rules of Nine Mens Morris / Merrills / Mill / Merels (mastersofgames.com)](https://www.mastersofgames.com/rules/morris-rules.htm?srsltid=AfmBOopb_phEilGCsxzu0B5c7vmMENtu1Ra-szEdqMWjEKIaIcSxIxIk)