

# KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY

Department of Computer Science and Technology

**Course Code: CSE 4110**

**Course Title: Artificial Intelligence Laboratory**

**Project Title: Quoridoor**

**Submitted to:**

## Md. Shahidul Salim

Lecturer

Department of Computer Science and Engineering

Khulna University of Engineering & Technology, Khulna

## Most. Kaniz Fatema Isha

Lecturer

Department of Computer Science and Engineering

Khulna University of Engineering & Technology, Khulna

**Submitted by:**

**Name: Imtiag Hossin Joy**

**Roll: 1907092**

**Name: Akib Nihal**

**Roll: 1907096**

Department of Computer Science and Engineering

Khulna University of Engineering & Technology, Khulna

# Introduction

## Background

Artificial Intelligence (AI) has made significant improvements in the development of intelligent systems capable of performing tasks that typically need human intelligence. In game development, AI techniques have been widely used to create dynamic gameplay experiences. This project involves the design and implementation of a Quoridor game using Python and the Pygame library, adding AI strategies such as the Minimax algorithm and Genetic Algorithms to enhance the gameplay.

The Quoridor game is a strategic board game that involves two players whose objective is to move their pawn from one side of the board to the opposite side. Players can also place walls to block their opponent's path. The game is quite simple in rules but the depth in strategy makes it an excellent board game for experimenting with different AI techniques.

Python is a powerful programming language, well-suited for both beginners and experienced developers. Its simplicity make it an ideal choice for implementing complex algorithms in a straightforward manner. Pygame, a set of Python modules designed for writing video games, provides the necessary tools for developing the graphical user interface and handling user inputs, making it an excellent choice for this project.

The Minimax algorithm is a recursive method used in decision-making and game theory to determine the optimal move for a player, assuming that the opponent is also playing optimally. In the context of Quoridor, the Minimax algorithm evaluates the possible moves of both players to minimize the possible loss in a worst-case scenario. By exploring the game tree up to a certain depth, the algorithm can choose the move that maximizes the player's chances of winning while minimizing the opponent's opportunities.

Genetic Algorithms (GAs) are optimization techniques inspired by the principles of natural selection and genetics. GAs are used to solve complex problems by evolving a population of candidate solutions over several generations. In this project, a Genetic Algorithm was employed to optimize certain aspects of the Quoridor game, such as finding the best weights that were used for evaluation function to find the appropriate move evaluation. By simulating the process of natur-

al evolution, GAs explore a vast search space and identify optimal or near-optimal weights that may not be immediately found out through traditional methods.

## Objectives

* Implementing the basic rules and mechanics of the Quoridor game.
* Developing a user-friendly graphical interface using Pygame.
* Integrating the Minimax algorithm to enable the AI to make optimal moves.
* Utilizing Genetic Algorithms to explore and optimize weights for evaluation function.

# Methodology

## Description of processes and methods used in the project

‘Board’ Class: The Board class in the game implementation serves as the core structure to manage and represent the state of the game. It handles the game board's appearance, the players' positions, wall placements, and various game mechanics. It has different attributes serving different purpose and they are:

* player1wall and player2wall: Track the number of walls each player has placed.
* current and opponent: Identify the current player and their opponent.
* players: Maintain the positions of the two players on the board.
* wall: Store the positions and orientations of the walls placed on the board.
* message and message\_time: Manage the display of messages to the players.

These attributes are used in different methods present in the class for the gameplay. There are several functions in the class serving distinct purpose for gameplay.

‘draw’ function: The purpose of the ‘draw’ method is to render the game board and its elements onto the Pygame window. It takes parameters like self, window. The self parameter refers to the instance of the Board class. It allows the method to access the attributes and other methods of the class. The window parameter s a Pygame surface object where the game elements (board, players, walls, messages) are drawn. This method does not return any value. Instead, it performs actions to draw the current state of the game on the provided window surface.

‘is\_valid\_cell’ function: The purpose of this method is to determine whether a given cell position (specified by row and col indices) is within the boundaries of the game board. This is useful for ensuring that moves and wall placements are within the valid playable area of the board. It takes parameters like self, row, col. Here the row and col are integers representing the row and col index of the cell to be checked. It returns a boolean value ‘True’ if the given row and col is within the boundaries of the board or otherwise ‘False’.

‘show\_message’ function: The purpose of the show\_message method is to display a message on the board for a specified duration. The message will be visible on the board for the given duration and then automatically cleared. It takes parameters like self, message, duration. Here the message is a string the represents the message to be displayed on the board and duration represents the time for how long the message should be displayed.

‘evaluate\_position’ function: The purpose of the evaluate\_position method is to calculate and return a numerical value representing the evaluation of the current game state. This evaluation helps determine how favorable the current position is for the player whose turn it is. It takes self as parameter and returns a float or integer representing the evaluation score of the current board position.

‘winner’ function: The purpose of the winner method is to check if any player has won the game by reaching their respective goal. It returns the color of the winning player if a win condition is met, otherwise it returns ‘None’. It takes self as the input parameter.

‘changeTurn’ function: The purpose of the changeTurn method is to toggle the turn between the two players. It switches the current player to the other player and updates the opponent attribute accordingly. It takes self as input parameter and returns nothing.

‘move\_player’ function: The purpose of the move\_player method is to update the position of a specified player on the board. It changes the player's coordinates to the specified row and column.

It takes input parameters like self, player which is an integer representing the player to be moved, row and col which represent the cell position where the player will be moved. This method does not return anything.

‘place\_wall’ function: The purpose of the place\_wall method is to add a wall to the board at the specified starting position (row, col) with the specified direction (dir) and color (color). The input parameters are self, row which is an integer representing the starting row position where the wall is to be placed, col which is an integer representing the starting column position where the wall is to be placed, dir which is an integer representing the direction of the wall (0 for horizontal, 1 for vertical), color which is a color value or identifier representing the color of the wall. This method returns nothing.

‘can\_move\_to’ function: The purpose of the can\_move\_to method is to determine if it's possible for a player to move to a specified target cell (row, col) on the board. It checks for the presence of walls and ensures that the target cell is within the boundaries of the board. It takes input parameters like self, players which is a tuple containing the current position (row, col) of the player, row which is an integer representing the row of the target cell, col which is an integer representing the column of the target cell. This method returns True if the player can move to the target cell (row, col), otherwise returns False.

‘can\_place\_wall’ function: The purpose of the can\_place\_wall method is to determine if it's valid to place a wall at a specified position on the board. It checks several conditions to ensure that the placement follows the game rules regarding board boundaries and existing walls. It takes input parameters like self, row representing the row where the wall is to be placed, col representing the column where the wall is to be placed and dir representing the direction of the wall (0 for horizontal, 1 for vertical).

‘get\_possible\_moves’ function: The purpose of the get\_possible\_moves method is to determine all possible legal moves for the current player on the board. It considers both movement options (up, down, left, right) and wall placement options, based on the game rules and current board state. It takes input parameters like self and player and returns a list of possible moves for the current

player. Each move is represented as a tuple: a move action: ("move", row, col), a wall placement action: ("wall", row, col, dir).

‘minimax’ function: The minimax function uses the minimax algorithm to determine the best move for the AI player. It explores possible future game states up to a certain depth, evaluating each state and choosing the move that leads to the best outcome for the AI while assuming the opponent will also play optimally. This allows the AI to make strategic decisions considering multiple steps ahead in the game. This function takes input parameters like position which is the current game state, depth which determines the depth of the search tree and max\_player which is a boolean indicating whether it's the maximizing player's turn (AI) or minimizing player's turn (human).

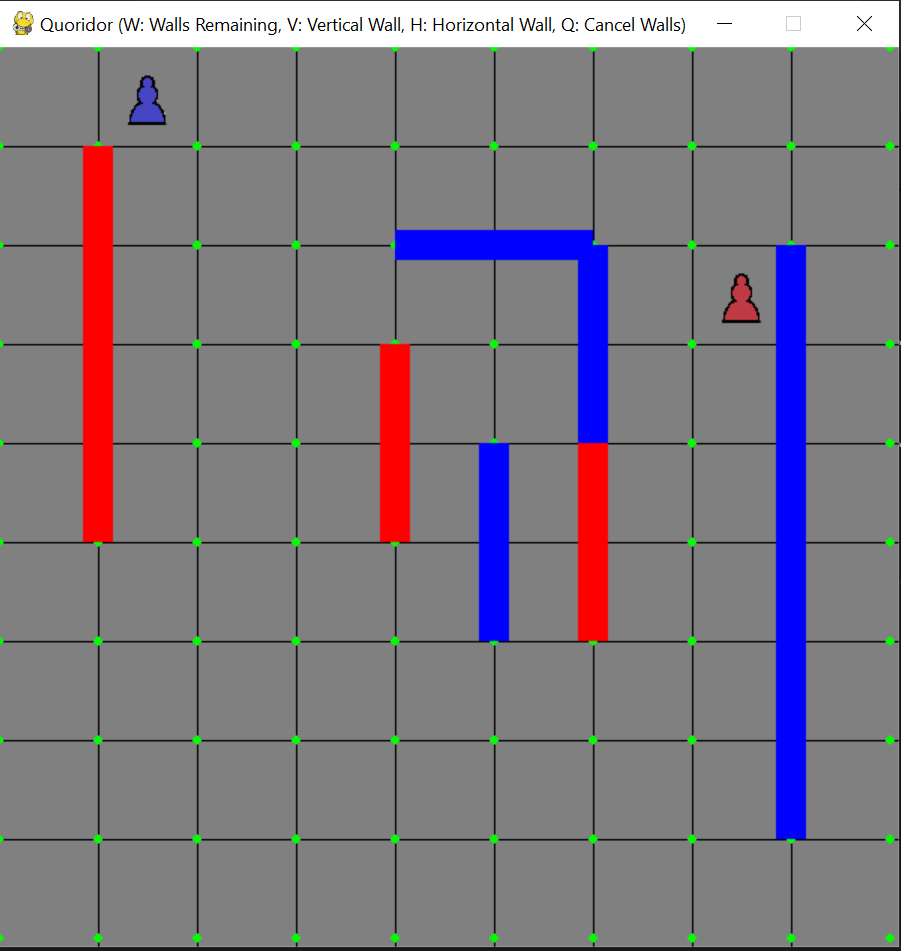
‘genetic\_algorithm’ function: The genetic algorithm is used to optimize the weight parameter in the board evaluation function. This weight balances the importance of different factors in evaluating a game state, such as player distance from the goal, wall placement, and available moves. By evolving this weight over multiple generations, the algorithm aims to find an optimal balance that improves the AI's decision-making. It takes input parameters like generations representing the number of generations to evolve the population, population\_size representing the number of individuals in each generation, lower\_bound which is the minimum value for the weight parameter, upper\_bound which is the maximum value for the weight parameter and mutation\_rate representing probability of mutation for each individual. The output of the method is best\_weight which is the optimized weight parameter for the evaluation function.

# Result

# The game that has been developed has several inputs and these inputs are handled by mouse and keyboard. All the important input methods are given in the caption of the game interface. Different input methods are given below:

Movement in board: The player can move in the board using the arrow keys. The player will move to the upper cell or lower cell or cell at right or cell at left from the current position by 1 cell.

Placing Walls in board: The player can place walls in two direction and they are vertical and horizontal. For placing vertical wall the player needs to press the ‘v’ key and then using the mouse the player just need to mark the cell from which the wall will be started and then another click in the direction players wants to give the wall. The placement method of horizontal wall is also same but the difference is that player have to press the ‘h’ key.



**Figure3.1:** Snapshot of the Quoridoor gameplay.

Remaining Wall Count: The player needs to press the ‘w’ key to view the remaining walls that can be placed in the board. The player will be given total 10 walls to place on the board.

Cancel Wall: The player can cancel the wall according to their will if they want after pressing the ‘h’ or ‘v’ key. For the accomplishment of this, the player needs to press the ‘q’ key.

# Discussion

The Quoridor game developed in this project successfully implements the core rules and mechanics of the original board game using Python and Pygame. The game allows two players to navigate their pawns across a 9x9 grid while strategically placing walls to hinder their opponent's progress.

The board is represented using a 2D grid where players and walls are placed based on their respective coordinates. Pygame is utilized for rendering the graphical interface, displaying the board, player pawns, and walls. Players can move their pawns orthogonally across the board and place walls to block their opponent's path. Movement and wall placement are validated using algorithms that ensure moves adhere to the game's rules, such as limiting wall placements and checking for valid paths. The game includes an AI opponent implemented using the minimax algorithm. This AI analyzes potential moves to determine the optimal strategy for both movement and wall placement, aiming to maximize its chances of winning while minimizing the opponent's opportunities. A heuristic evaluation function is employed to assess the current game state and guide decision-making for both the AI and player moves. This function incorporates various factors such as player distances to victory, progress towards goals, number of available moves, and wall placements. A genetic algorithm is utilized to optimize the weights used in the heuristic evaluation function dynamically. By evolving these weights over successive generations, the game AI improves its ability to evaluate board positions effectively, enhancing its strategic decision-making capabilities.

The implemented game provides a balanced gameplay experience, offering strategic depth through the combination of movement tactics and wall placements. Players can enjoy competitive matches against the AI. The Pygame-based graphical user interface enhances the user experience with board elements, player turns, and game messages. The interface includes features such as mouse-click interactions for move and wall placement, improving accessibility and usability.

# Future Work

AI Optimization and Expansion:

* Further refinement of the heuristic evaluation function through continued experimentation with genetic algorithms to optimize weights dynamically.
* Implementing more sophisticated AI strategies, such as reinforcement learning algorithms, to adapt and learn from gameplay experiences.

User Interface Enhancements:

* Incorporating enhanced graphics, animations for movements, and intuitive user interface elements to improve player engagement.
* Adding accessibility options, such as customizable color schemes and interface layouts.

Gameplay Features:

* Multiplayer Functionality: Implementing network multiplayer capabilities to enable competitive matches between players across different devices.

Performance Optimization:

* Optimizing algorithms for faster decision-making and smoother gameplay.

# Conclusion

The development of the Quoridor game using Python and Pygame has successfully showed the integration of fundamental AI techniques within a strategic board game context. By implementing the core mechanics of the game and incorporating AI strategies such as the Minimax algorithm and Genetic Algorithms a dynamic and engaging gameplay experience have been created.

The Minimax algorithm effectively provides the AI with the capability to make optimal decisions, considering multiple future moves to outmaneuver the opponent. The Genetic Algorithm has further enhanced the AI by optimizing the evaluation function, leading to improved decision-making that adapts to different game scenarios. This project showcases the power of Python and Pygame in game development, allowing for straightforward implementation of complex algorithms and creating an intuitive graphical interface. The game’s design ensures that both human and AI players can enjoy a challenging and strategic experience, adhering closely to the original Quoridor board game's rules.

Future work on this project could involve refining the AI further, incorporating advanced techniques such as reinforcement learning, enhancing the graphical interface, and adding multiplayer capabilities. These improvements would not only enrich the gameplay but also demonstrate more advanced applications of AI in game development.

Overall, this project highlights the potential of AI in creating intelligent, strategic game experiences and provides a solid foundation for further exploration and development in the field of game AI.

# Contributions

* 1907092: User Interface, Genetic Algorithm
* 1907096: User Interface, Minmax Algorithm

# References

1. Mertens, Peter JC. "A quoridor-playing agent." *Bachelor Thesis, Department of Knowledge Engineering, Maastricht University* (2006).
2. Glendenning, Lisa. "Mastering quoridor." *Bachelor Thesis, Department of Computer Science, The University of New Mexico* (2005).
3. https://www.pygame.org.docs/