

|  |
| --- |
| **RAJALAKSHMI INSTITUTE OF TECHNOLOGY** |
| (An Autonomous Institution, Affiliated to Anna University, Chennai) |

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**ACADEMIC YEAR 2025 - 2026**

**SEMESTER III**

**ARTIFICIAL INTELLIGENCE LABORATORY**

**MINI PROJECT REPORT**

|  |  |
| --- | --- |
| **REGISTER NUMBER** | 2117240070027 |
| **NAME** | ASHWIN KUMAAR.B |
| **PROJECT TITLE** | connect 4 game using minimax algorithm |
| **DATE OF SUBMISSION** |  |
| **FACULTY IN-CHARGE** | **Mrs. M. Divya** |

**Signature of Faculty In-charge**

**INTRODUCTION**

Artificial Intelligence (AI) enables machines to make decisions similar to humans. Game-playing is one of AI’s earliest and most practical applications. This project implements the classic Connect Four game where the player competes against an AI opponent. The AI uses the Minimax algorithm with Alpha–Beta pruning to make optimal decisions, demonstrating how AI evaluates and predicts outcomes.

**PROBLEM STATEMENT**

Design a Connect Four game where a human player competes against a computer that makes intelligent, strategic moves using AI.

**THEORETICAL BACKGROUND**

The **Minimax algorithm** is a decision-making algorithm used in turn-based games. It simulates all possible moves, assuming both players play optimally, and selects the move with the best guaranteed outcome.  
**Alpha–Beta pruning** reduces computation by skipping branches that don’t affect the final decision.  
This combination provides a balance between intelligence and efficiency, making it ideal for small board games like Connect Four.

**ALGORITHM EXPLANATION WITH EXAMPLE**

1. Generate all valid moves.
2. Simulate each move recursively until a terminal state or depth limit
3. Assign scores (+ for AI, – for player).
4. Backtrack and choose the move with the highest score (AI) or lowest (player).

**Example:**  
If AI can win in one move, it selects that column immediately. Otherwise, it blocks the opponent or builds a better position.

**IMPLEMENTATION AND CODE**

class ConnectFour:

def \_\_init\_\_(self, rows=6, cols=7):

self.rows = rows

self.cols = cols

self.board = [[0 for \_ in range(cols)] for \_ in range(rows)]

self.current\_player = 1

self.game\_over = False

self.winner = None

def print\_board(self):

print("\n" + "=" \* 30)

print(" CONNECT FOUR GAME")

print("=" \* 30)

print(" " + " ".join(str(i) for i in range(self.cols)))

print(" " + "-" \* (self.cols \* 2 - 1))

for row in range(self.rows):

print("| ", end="")

for col in range(self.cols):

if self.board[row][col] == 0:

print(".", end=" ")

elif self.board[row][col] == 1:

print("X", end=" ")

else:

print("O", end=" ")

print("|")

print(" " + "-" \* (self.cols \* 2 - 1))

print()

def is\_valid\_move(self, col):

return 0 <= col < self.cols and self.board[0][col] == 0

def get\_valid\_moves(self):

return [col for col in range(self.cols) if self.is\_valid\_move(col)]

def make\_move(self, col, player):

if not self.is\_valid\_move(col):

return False

for row in range(self.rows-1, -1, -1):

if self.board[row][col] == 0:

self.board[row][col] = player

return True

return False

def check\_winner(self, player):

for row in range(self.rows):

for col in range(self.cols - 3):

if all(self.board[row][col+i] == player for i in range(4)):

return True

for row in range(self.rows - 3):

for col in range(self.cols):

if all(self.board[row+i][col] == player for i in range(4)):

return True

for row in range(self.rows - 3):

for col in range(self.cols - 3):

if all(self.board[row+i][col+i] == player for i in range(4)):

return True

for row in range(self.rows - 3):

for col in range(3, self.cols):

if all(self.board[row+i][col-i] == player for i in range(4)):

return True

return False

def is\_board\_full(self):

return all(self.board[0][col] != 0 for col in range(self.cols))

def evaluate\_window(self, window, player):

opponent = 3 - player

score = 0

if window.count(player) == 4:

score += 100

elif window.count(player) == 3 and window.count(0) == 1:

score += 5

elif window.count(player) == 2 and window.count(0) == 2:

score += 2

if window.count(opponent) == 3 and window.count(0) == 1:

score -= 4

return score

def evaluate\_board(self, player):

score = 0

opponent = 3 - player

center\_col = self.cols // 2

center\_array = [self.board[row][center\_col] for row in range(self.rows)]

center\_count = center\_array.count(player)

score += center\_count \* 3

for row in range(self.rows):

for col in range(self.cols - 3):

window = [self.board[row][col+i] for i in range(4)]

score += self.evaluate\_window(window, player)

for row in range(self.rows - 3):

for col in range(self.cols):

window = [self.board[row+i][col] for i in range(4)]

score += self.evaluate\_window(window, player)

for row in range(self.rows - 3):

for col in range(self.cols - 3):

window = [self.board[row+i][col+i] for i in range(4)]

score += self.evaluate\_window(window, player)

for row in range(self.rows - 3):

for col in range(3, self.cols):

window = [self.board[row+i][col-i] for i in range(4)]

score += self.evaluate\_window(window, player)

return score

def get\_next\_open\_row(self, col):

for row in range(self.rows-1, -1, -1):

if self.board[row][col] == 0:

return row

return -1

def minimax(self, depth, alpha, beta, maximizing\_player):

valid\_moves = self.get\_valid\_moves()

is\_terminal = self.check\_winner(1) or self.check\_winner(2) or self.is\_board\_full()

if depth == 0 or is\_terminal:

if self.check\_winner(2):

return None, 1000000000

elif self.check\_winner(1):

return None, -1000000000

elif self.is\_board\_full():

return None, 0

else:

return None, self.evaluate\_board(2)

if maximizing\_player:

value = -float('inf')

best\_column = valid\_moves[0] if valid\_moves else 0

for col in valid\_moves:

row = self.get\_next\_open\_row(col)

self.board[row][col] = 2

new\_score = self.minimax(depth-1, alpha, beta, False)[1]

self.board[row][col] = 0

if new\_score > value:

value = new\_score

best\_column = col

alpha = max(alpha, value)

if alpha >= beta:

break

return best\_column, value

else:

value = float('inf')

best\_column = valid\_moves[0] if valid\_moves else 0

for col in valid\_moves:

row = self.get\_next\_open\_row(col)

self.board[row][col] = 1

new\_score = self.minimax(depth-1, alpha, beta, True)[1]

self.board[row][col] = 0

if new\_score < value:

value = new\_score

best\_column = col

beta = min(beta, value)

if alpha >= beta:

break

return best\_column, value

def get\_ai\_move(self, depth=4):

print("AI is thinking...")

for col in self.get\_valid\_moves():

row = self.get\_next\_open\_row(col)

self.board[row][col] = 2

if self.check\_winner(2):

self.board[row][col] = 0

return col

self.board[row][col] = 0

for col in self.get\_valid\_moves():

row = self.get\_next\_open\_row(col)

self.board[row][col] = 1

if self.check\_winner(1):

self.board[row][col] = 0

return col

self.board[row][col] = 0

column, value = self.minimax(depth, -float('inf'), float('inf'), True)

return column

def play\_game(self):

print("Welcome to Connect Four!")

print("You are X, AI is O")

print("Enter column numbers (0-6) to make your moves")

while not self.game\_over:

self.print\_board()

if self.current\_player == 1:

try:

col = int(input("Your turn! Enter column (0-6): "))

if not self.is\_valid\_move(col):

print("Invalid move! Column is full or out of range.")

continue

self.make\_move(col, 1)

if self.check\_winner(1):

self.print\_board()

print("🎉 Congratulations! You won!")

self.game\_over = True

self.winner = 1

except ValueError:

print("Please enter a valid number!")

continue

else:

col = self.get\_ai\_move()

self.make\_move(col, 2)

print(f"AI plays in column {col}")

if self.check\_winner(2):

self.print\_board()

print("🤖 AI wins! Better luck next time!")

self.game\_over = True

self.winner = 2

if self.is\_board\_full() and not self.game\_over:

self.print\_board()

print("It's a draw!")

self.game\_over = True

self.current\_player = 3 - self.current\_player

def main():

game = ConnectFour()

game.play\_game()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**OUTPUT**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

* Text-based board display with columns (0–6).
* Player inputs column; AI responds after evaluating.
* Displays win, lose, or draw messages.

**RESULTS AND FUTURE ENHANCEMENT**

The AI successfully competes against the player with intelligent decisions.  
Future improvements:

* Add GUI using Tkinter or Pygame.
* Adjustable difficulty levels.
* Use advanced AI like Reinforcement Learning for adaptive play.

|  |  |
| --- | --- |
| **Git Hub Link of the project and report** | **https://github.com/ashwin-kumaar-b/ai\_miniproject** |

**REFERENCES**

* Stuart Russell & Peter Norvig, *Artificial Intelligence: A Modern Approach*, Pearson.
* GeeksforGeeks – “Minimax Algorithm in Game Theory.”
* Wikipedia – “Connect Four.”
* Towards Data Science – “Alpha–Beta Pruning Explained.”
* TutorialsPoint – “AI – Game Playing Techniques.”