Title Page

Noughts and Crosses with Alpha-Beta Pruning

Name:Briteshwar kumar

Roll No.202401100300091

2. Introduction

This report presents an implementation of Noughts and Crosses (Tic-Tac-Toe) using the Minimax algorithm with Alpha-Beta Pruning. The AI efficiently selects optimal moves against a human player.

3. Methodology

The game is built using Python and follows a structured approach:

- The board is represented as a 3x3 grid.

- The Minimax algorithm with Alpha-Beta Pruning determines the best possible move for the AI.

- The AI plays as 'X', while the human player plays as 'O'.

- The game checks for a winner after each move.

- The game continues until there is a winner or the board is full.

4. Code Typed

```python

import math

def print\_board(board):

for row in board:

print(" ".join(row))

print()

def check\_winner(board):

for row in board:

if row[0] == row[1] == row[2] and row[0] != ' ':

return row[0]

for col in range(3):

if board[0][col] == board[1][col] == board[2][col] and board[0][col] != ' ':

return board[0][col]

if board[0][0] == board[1][1] == board[2][2] and board[0][0] != ' ':

return board[0][0]

if board[0][2] == board[1][1] == board[2][0] and board[0][2] != ' ':

return board[0][2]

return None

def is\_full(board):

return all(board[r][c] != ' ' for r in range(3) for c in range(3))

def minimax(board, depth, alpha, beta, is\_maximizing):

winner = check\_winner(board)

if winner == 'X':

return 1

elif winner == 'O':

return -1

elif is\_full(board):

return 0

if is\_maximizing:

max\_eval = -math.inf

for r in range(3):

for c in range(3):

if board[r][c] == ' ':

board[r][c] = 'X'

eval = minimax(board, depth + 1, alpha, beta, False)

board[r][c] = ' '

max\_eval = max(max\_eval, eval)

alpha = max(alpha, eval)

if beta <= alpha:

break

return max\_eval

else:

min\_eval = math.inf

for r in range(3):

for c in range(3):

if board[r][c] == ' ':

board[r][c] = 'O'

eval = minimax(board, depth + 1, alpha, beta, True)

board[r][c] = ' '

min\_eval = min(min\_eval, eval)

beta = min(beta, eval)

if beta <= alpha:

break

return min\_eval

def best\_move(board):

best\_val = -math.inf

move = (-1, -1)

for r in range(3):

for c in range(3):

if board[r][c] == ' ':

board[r][c] = 'X'

move\_val = minimax(board, 0, -math.inf, math.inf, False)

board[r][c] = ' '

if move\_val > best\_val:

best\_val = move\_val

move = (r, c)

return move

def play\_game():

board = [[' ' for \_ in range(3)] for \_ in range(3)]

print("Noughts and Crosses: AI (X) vs. Human (O)")

print\_board(board)

while True:

if is\_full(board) or check\_winner(board):

break

row, col = best\_move(board)

board[row][col] = 'X'

print("AI chooses:")

print\_board(board)

if check\_winner(board) or is\_full(board):

break

while True:

try:

user\_row, user\_col = map(int, input("Enter row and column (0-2) separated by space: ").split())

if board[user\_row][user\_col] == ' ':

board[user\_row][user\_col] = 'O'

break

else:

print("Invalid move, try again.")

except (ValueError, IndexError):

print("Invalid input, enter numbers between 0 and 2.")

print\_board(board)

winner = check\_winner(board)

if winner:

print(f"{winner} wins!")

else:

print("It's a draw!")

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

play\_game()

```

5. ScreenShots Output photo pasted

