**TOPIC** : NOUGHTS AND CROSSES WITH

ALPHA-BETA PRUNING

**NAME : GAVENDRA PACHAHRA**

**UNIV. ROLL NO. : 202401100300111**

**BRANCH : CSE-AI**

**SUBJECT : INTRODUCTION TO AI**

**SUBJECT CODE : AI101B**

**Introduction** :

Noughts and Crosses (Tic-Tac-Toe) is a classic two-player game where players take turns marking a 3×3 grid with 'X' or 'O'. The objective is to form a row, column, or diagonal with three matching symbols. This project implements an AI-powered version using the Minimax algorithm with Alpha-Beta Pruning, ensuring optimal gameplay.

**Methodology :**

Game Representation: The board is implemented as a 3×3 matrix.

AI Decision-Making: Uses the Minimax algorithm with Alpha-Beta Pruning to determine the best move.

Optimization: Alpha-Beta Pruning speeds up decision-making by eliminating unnecessary evaluations.

User Interaction: The player inputs their move while the AI calculates the optimal response.

CODE :

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]

if all(board[row][col] != ' ' for row in range(3) for col in range(3)):

return 'Draw'

return None

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

winner = check\_winner(board)

if winner == 'X':

return 1

elif winner == 'O':

return -1

elif winner == 'Draw':

return 0

if is\_maximizing:

max\_eval = -math.inf

for row in range(3):

for col in range(3):

if board[row][col] == ' ':

board[row][col] = 'X'

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

board[row][col] = ' '

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

alpha = max(alpha, eval)

if beta <= alpha:

break

return max\_eval

else:

min\_eval = math.inf

for row in range(3):

for col in range(3):

if board[row][col] == ' ':

board[row][col] = 'O'

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

board[row][col] = ' '

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 row in range(3):

for col in range(3):

if board[row][col] == ' ':

board[row][col] = 'X'

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

board[row][col] = ' '

if move\_val > best\_val:

best\_val = move\_val

move = (row, col)

return move

def play\_game():

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

while True:

print\_board(board)

winner = check\_winner(board)

if winner:

print("Winner:", winner)

break

row, col = best\_move(board)

board[row][col] = 'X'

print("AI plays X at:", row, col)

print\_board(board)

winner = check\_winner(board)

if winner:

print("Winner:", winner)

break

player\_row = int(input("Enter row (0-2): "))

player\_col = int(input("Enter col (0-2): "))

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

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

else:

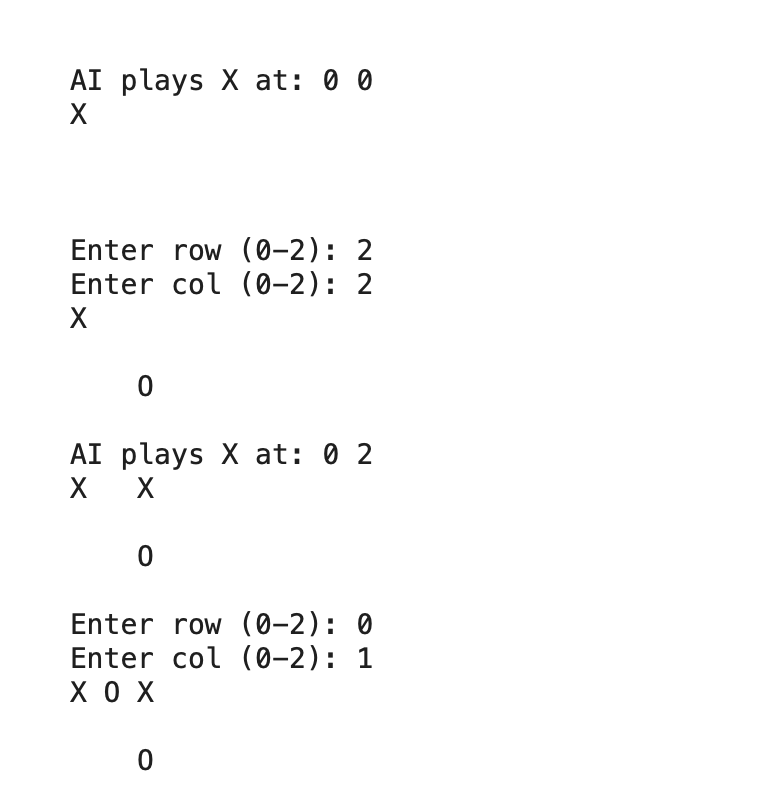
print("Invalid move, try again.")

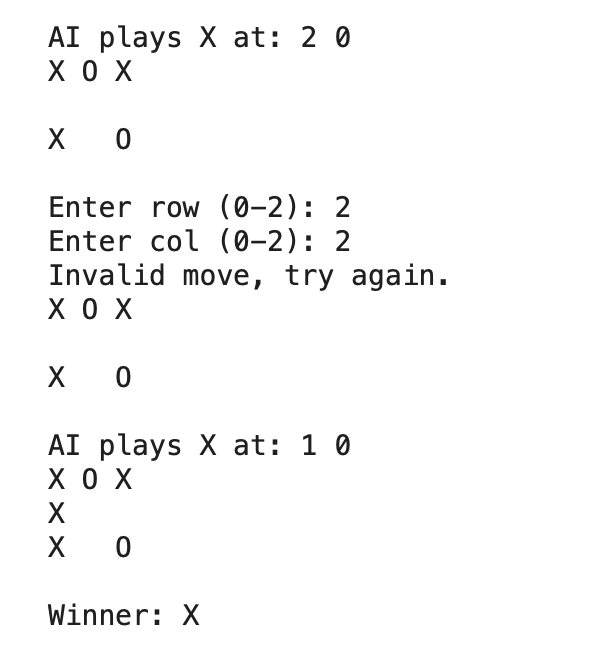
continue

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

play\_game()

OUTPUTS :





**Conclusion :**

The implementation of Noughts and Crosses using the Minimax algorithm with Alpha-Beta Pruning results in an AI that plays optimally and efficiently. Alpha-Beta Pruning significantly enhances performance by reducing the number of nodes evaluated in the game tree. The AI is unbeatable, ensuring the best possible move in every situation. This project successfully demonstrates the application of game theory and decision-making algorithms in AI-driven gameplay. Future improvements could include graphical user interface integration and different difficulty levels for varied gameplay experiences.

**THANKING YOU**