Slide 1: Title Page

- **Project Title**: **Noughts and Crosses with Alpha-Beta Pruning**
- **Your Name**: Shashi Ranjan
- **Course Name-** Introduction to AI
- **Instructor Name**: Bhavna Bansal
- **Date**: 10/03/2025

Slide 2: Introduction

1. **Overview of the Game**:
   - *Noughts and Crosses* (Tic-Tac-Toe) is a classic 2-player game where players take turns to place their marks ("X" and "O") on a 3x3 grid.
   - The goal is to get three of one's marks in a row—horizontally, vertically, or diagonally**.**
2. **AI Implementation**:
   - In this project, an AI is developed to play against a human player.
   - The AI uses the **Alpha-Beta Pruning** technique to optimize its decision-making process.
3. **Objective**:
   - To demonstrate how Alpha-Beta Pruning optimizes the **Minimax algorithm**, reducing the computation time and improving performance.
4.
5.

Slide 3: Methodology

1. **Game Representation**:
   - The game board is represented as a 1D array with 9 positions.
   - Values in the array represent:
     - `0` = Empty position
     - `1` = "X" (AI's move)
     - `-1` = "O" (Player's move)

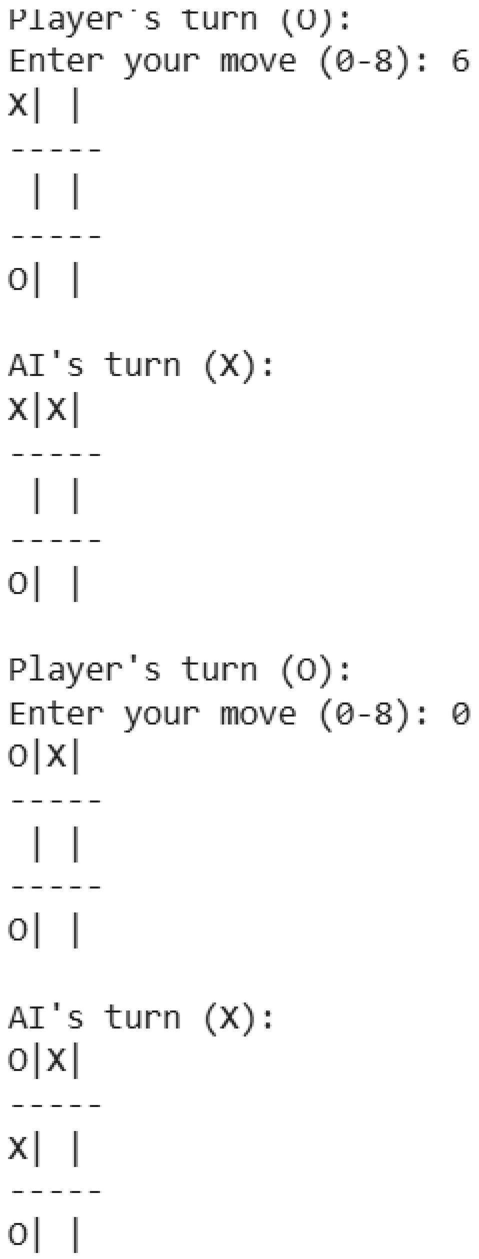
2. **Minimax Algorithm**:
    - The AI evaluates all possible moves to determine the optimal one, assuming both players play optimally.
3. **Alpha-Beta Pruning**:
    - Alpha-Beta Pruning is used to enhance the Minimax algorithm by reducing the number of branches it needs to evaluate.
    - It prunes (cuts off) branches that will not affect the final decision
4. **Code Overview**:
    - Here is the key Python code that implements the Alpha-Beta Pruning logic for Tic-Tac-Toe.

Code Implementation

- **Code Overview**:
    - Here is the key Python code that implements the Alpha-Beta Pruning logic for

```
import math



# Board representation: 0 = empty, 1 = X (AI), -1 = O (Player)

board = [0] * 9



# Check for winner

def check_winner(board):

  win_cond = [(0, 1, 2), (3, 4, 5), (6, 7, 8), (0, 3, 6), (1, 4, 7), (2, 5, 8), (0, 4, 8), (2, 4, 6)]

  for a, b, c in win_cond:

    if board[a] == board[b] == board[c] and board[a] != 0:

      return board[a]

  return 0



# Check if the game is over
```

```
def is_game_over(board):
    return check_winner(board) != 0 or all(x != 0 for x in board)


# Alpha-Beta Pruning: Minimax algorithm
def alpha_beta(board, depth, alpha, beta, maximizing_player):
    winner = check_winner(board)
    if winner != 0:
        return winner * (10 - depth)  # Positive score for X, negative for O
    if is_game_over(board):
        return 0  # Draw

    valid_moves = [i for i, x in enumerate(board) if x == 0]


    if maximizing_player:  # AI's turn (X)
        max_eval = -math.inf
        for move in valid_moves:
            board[move] = 1  # X plays
            eval = alpha_beta(board, depth + 1, alpha, beta, False)
            board[move] = 0
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
        return max_eval
    else:  # Player's turn (O)
        min_eval = math.inf
        for move in valid_moves:
            board[move] = -1  # O plays
            eval = alpha_beta(board, depth + 1, alpha, beta, True)
            board[move] = 0
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break
        return min_eval

# Get the best move for the AI
def best_move(board):
    best_score = -math.inf
    move = -1
    for i in range(9):
        if board[i] == 0:
            board[i] = 1  # AI is X
            score = alpha_beta(board, 0, -math.inf, math.inf, False)
            board[i] = 0
            if score > best_score:
                best_score = score
                move = i
    return move

# Display the board
def print_board(board):
    symbols = [' ', 'X', 'O']
    for i in range(3):

print(f"{symbols[board[i*3]]}|{symbols[board[i*3+1]]}|{symbols[board[i*3+2]]}")
        if i < 2: print("-----")
    print()

# Play the game
def play_game():
    turn = 1  # X starts
    while not is_game_over(board):
        print_board(board)
        if turn == 1:
            print("AI's turn (X):")
            move = best_move(board)
        else:
            print("Player's turn (O):")
            move = int(input("Enter your move (0-8): "))

        board[move] = turn
        turn *= -1  # Switch player

    print_board(board)
    winner = check_winner(board)
    if winner == 1:
        print("AI wins!")
    elif winner == -1:
        print("Player wins!")
    else:
        print("It's a draw!")

# Start the game
play_game()
```



```
Player's turn (O):
Enter your move (0-8): 6
X| |
-----
 | |
-----
O| |

AI's turn (X):
X|X|
-----
 | |
-----
O| |

Player's turn (O):
Enter your move (0-8): 0
O|X|
-----
 | |
-----
O| |

AI's turn (X):
O|X|
-----
X| |
-----
O| |
```