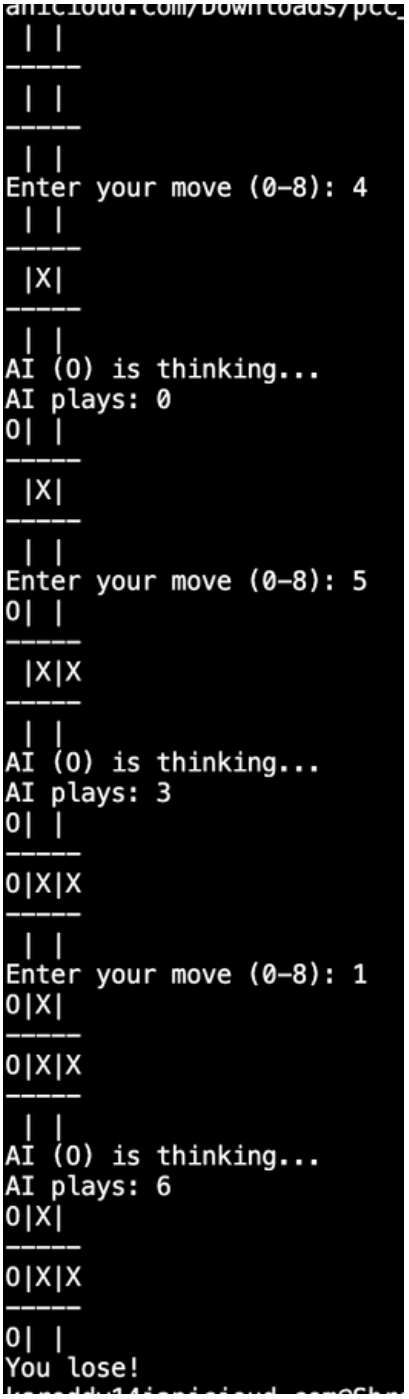
GAME SEARCH:

```
class TicTacToe:
    def __init__(self):
        self.board = [' '] * 9
        self.current_player = 'X'

    def print_board(self):
        for i in range(0, 9, 3):
            print("|".join(self.board[i:i+3]))
            if i < 6:
                print("-----")

    def is_winner(self, player):
        # Check rows
        for i in range(0, 9, 3):
            if all(self.board[j] == player for j in range(i, i+3)):
                return True

        # Check columns
        for i in range(3):
            if all(self.board[j] == player for j in range(i, 9, 3)):
                return True

        # Check diagonals
        if all(self.board[i] == player for i in [0, 4, 8]):
            return True
        if all(self.board[i] == player for i in [2, 4, 6]):
            return True

        return False

    def is_full(self):
        return ' ' not in self.board

    def is_game_over(self):
        return self.is_winner('X') or self.is_winner('O') or self.is_full()

    def get_available_moves(self):
        return [i for i, v in enumerate(self.board) if v == ' ']

    def make_move(self, move):
        self.board[move] = self.current_player
        self.current_player = 'O' if self.current_player == 'X' else 'X'
```

```
def undo_move(self, move):
        self.board[move] = ' '
        self.current_player = 'O' if self.current_player == 'X' else 'X
def minimax(board, maximizing_player):
    if board.is_game_over():
        if board.is_winner('X'):
            return -1
        elif board.is_winner('O'):
            return 1
        else:
            return 0
    if maximizing_player:
        max_eval = float('-inf')
        for move in board.get_available_moves():
            board.make_move(move)
            eval = minimax(board, False)
            board.undo_move(move)
            max_eval = max(max_eval, eval)
        return max_eval
    else:
        min_eval = float('inf')
        for move in board.get_available_moves():
            board.make_move(move)
            eval = minimax(board, True)
            board.undo_move(move)
            min_eval = min(min_eval, eval)
        return min_eval
def get_best_move(board):
    best_move = None
    best_eval = float('-inf')
    for move in board.get_available_moves():
        board.make_move(move)
        eval = minimax(board, False)
        board.undo_move(move)
        if eval > best_eval:
            best_eval = eval
            best_move = move
    return best_move
# Play the Game
game = TicTacToe()
while not game.is_game_over():
    game.print_board()
```

```
if game.current_player == 'X':
 try:
 move = int(input("Enter your move (0-8): "))
 except ValueError:
 print("Invalid input! Enter a number 0-8.")
 continue

 if move not in game.get_available_moves():
 print("Invalid move! Try again.")
 continue

 game.make_move(move)

 else:
 print("AI (O) is thinking...")
 move = get_best_move(game)
 print(f"AI plays: {move}")
 game.make_move(move)

game.print_board()

if game.is_winner('X'):
 print("You win!")
elif game.is_winner('O'):
 print("You lose!")
else:
 print("It's a draw!")
```



ALPHABETA PRUNING:

```
class TicTacToe:
    def __init__(self):
        self.board = [' '] * 9
        self.current_player = 'X'

    def print_board(self):
        for i in range(0, 9, 3):
            print(" " + " | ".join(self.board[i:i+3]) + " ")
            if i < 6:
                print("---+---+---")

    def is_winner(self, player):
        # Rows
        for i in range(0, 9, 3):
            if self.board[i] == self.board[i+1] == self.board[i+2] == player:
                return True

        # Columns
        for i in range(3):
            if self.board[i] == self.board[i+3] == self.board[i+6] == player:
                return True

        # Diagonals
        if self.board[0] == self.board[4] == self.board[8] == player:
            return True
        if self.board[2] == self.board[4] == self.board[6] == player:
            return True

        return False

    def is_full(self):
        return ' ' not in self.board

    def is_game_over(self):
        return self.is_winner('X') or self.is_winner('O') or self.is_full()

    def get_available_moves(self):
        return [i for i, v in enumerate(self.board) if v == ' ']

    def make_move(self, move):
        if 0 <= move < 9 and self.board[move] == ' ':
            self.board[move] = self.current_player
            self.current_player = 'O' if self.current_player == 'X' else 'X'
```

```
 return True
 return False

 def undo_move(self, move):
 self.board[move] = ' '
 self.current_player = 'O' if self.current_player == 'X' else 'X'


def evaluate(board):
 if board.is_winner('O'): # AI = O (maximizer)
 return 1
 if board.is_winner('X'): # Human = X (minimizer)
 return -1
 return 0


def minimax(board, depth, alpha, beta, maximizing_player):
 if board.is_game_over():
 return evaluate(board)

 if maximizing_player: # AI is maximizing
 max_eval = float('-inf')
 for move in board.get_available_moves():
 board.make_move(move)
 eval = minimax(board, depth + 1, alpha, beta, False)
 board.undo_move(move)
 max_eval = max(max_eval, eval)
 alpha = max(alpha, eval)
 if beta <= alpha:
 break # Beta cutoff
 return max_eval

 else: # Human is minimizing
 min_eval = float('inf')
 for move in board.get_available_moves():
 board.make_move(move)
 eval = minimax(board, depth + 1, alpha, beta, True)
 board.undo_move(move)
 min_eval = min(min_eval, eval)
 beta = min(beta, eval)
 if beta <= alpha:
 break # Alpha cutoff
 return min_eval
```

```
def main():
 game = TicTacToe()

 while not game.is_game_over():
 game.print_board()

 if game.current_player == 'X':
 while True:
 try:
 move = int(input("Your move (0-8): "))
 if game.make_move(move):
 break
 else:
 print("Invalid or occupied position.")
 except ValueError:
 print("Please enter a number 0-8.")
 else:
 print("AI (O) is thinking...")
 move = get_best_move(game)
 print(f"AI plays at position {move}")
 game.make_move(move)

 game.print_board()

 if game.is_winner('X'):
 print("You win! 🎉")
 elif game.is_winner('O'):
 print("AI wins! 🤖")
 else:
 print("Draw!")


if __name__ == "__main__":
 main()
```