# Define constants for players

EMPTY = 0

AI\_PLAYER = 1

HUMAN\_PLAYER = -1

def generate\_moves(board):

"""Generate all possible moves (empty cells) on the board."""

moves = []

for i in range(3):

for j in range(3):

if board[i][j] == EMPTY:

moves.append((i, j))

return moves

def evaluate(board):

"""Evaluate the desirability of the current game state."""

# Implement your heuristic evaluation function here

pass

def minimax(board, depth, maximizing\_player):

"""Minimax algorithm with optional Alpha-Beta Pruning."""

# Implement Minimax algorithm with or without Alpha-Beta Pruning

pass

def ai\_move(board):

"""Get the AI's move using Minimax algorithm."""

move = minimax(board, depth=2, maximizing\_player=True)

return move

def human\_move(board):

"""Get the human player's move."""

# Implement input handling to get the human player's move

pass

def print\_board(board):

"""Print the current game board."""

for row in board:

print(" ".join(map(str, row)))

def main():

board = [[EMPTY, EMPTY, EMPTY],

[EMPTY, EMPTY, EMPTY],

[EMPTY, EMPTY, EMPTY]]

print("Welcome to Tic-Tac-Toe!")

while True:

# AI's turn

ai\_row, ai\_col = ai\_move(board)

board[ai\_row][ai\_col] = AI\_PLAYER

print("AI's move:")

print\_board(board)

# Check for win or draw

# Implement this part

# Human's turn

human\_row, human\_col = human\_move(board)

board[human\_row][human\_col] = HUMAN\_PLAYER

print("Your move:")

print\_board(board)

# Check for win or draw

# Implement this part

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

main()