```
import math



# Constants

PLAYER_X = 'X'

PLAYER_O = 'O'

EMPTY = ' '



# Game board

board = [[EMPTY for _ in range(3)] for _ in range(3)]



# Functions to print the board
```

```
def print_board():
  for row in board:
      print("|".join(row))
      print("-" * 5)



# Check if the game is over
def is_game_over():
  # Check rows, columns, and diagonals for a win
  for row in range(3):
      if board[row][0] == board[row][1] == board[row][2] != EMPTY:
          return True
```

```
for col in range(3):
    if board[0][col] == board[1][col] == board[2][col] != EMPTY:
        return True
if board[0][0] == board[1][1] == board[2][2] != EMPTY:
    return True
if board[0][2] == board[1][1] == board[2][0] != EMPTY:
    return True
# Check if there are any empty spaces
for row in board:
    for cell in row:
        if cell == EMPTY:
```

```
            return False
    return True # Draw if no empty spaces


# Evaluate the board to return a score
def evaluate():
    for row in range(3):
        if board[row][0] == board[row][1] == board[row][2] != EMPTY:
            return 1 if board[row][0] == PLAYER_O else -1
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != EMPTY:
            return 1 if board[0][col] == PLAYER_O else -1
```

```
  if board[0][0] == board[1][1] == board[2][2] != EMPTY:
      return 1 if board[0][0] == PLAYER_O else -1
  if board[0][2] == board[1][1] == board[2][0] != EMPTY:
      return 1 if board[0][2] == PLAYER_O else -1
  return 0 # Draw



# Minimax algorithm with Alpha-Beta Pruning
def minimax(depth, is_maximizing, alpha, beta):
  score = evaluate()



  # If the game is over, return the score
```

```
if score == 1 or score == -1:
    return score



if is_game_over():
    return 0 # Draw



if is_maximizing:
    max_eval = -math.inf
    for row in range(3):
        for col in range(3):
            if board[row][col] == EMPTY:
```

```
            board[row][col] = PLAYER_O
            eval = minimax(depth + 1, False, alpha, beta)
            board[row][col] = EMPTY
            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] == EMPTY:
            board[row][col] = PLAYER_X
            eval = minimax(depth + 1, True, alpha, beta)
            board[row][col] = EMPTY
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break
return min_eval
```

```
# Find the best move for AI

def best_move():

  best_val = -math.inf

  move = (-1, -1)



  for row in range(3):

     for col in range(3):

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

           board[row][col] = PLAYER_O

           move_val = minimax(0, False, -math.inf, math.inf)

           board[row][col] = EMPTY
```

```
            if move_val > best_val:
                best_val = move_val
                move = (row, col)

  return move

# Get human player's move
def human_move():
  while True:
    try:
      row, col = map(int, input("Enter your move (row, col) between 0-2: ").split())
```

```
            if board[row][col] == EMPTY:
                board[row][col] = PLAYER_X
                break
            else:
                print("Cell is already occupied. Try again.")
        except (ValueError, IndexError):
            print("Invalid move. Please enter row and col between 0 and 2.")


# Main game loop
def play_game():
  print("Welcome to Tic-Tac-Toe!")
```

```
print_board()



while not is_game_over():
    # Human move
    human_move()
    print_board()


    if is_game_over():
        break


    # AI move
```

```
        print("AI's move:")
        ai_move = best_move()
        board[ai_move[0]][ai_move[1]] = PLAYER_O
        print_board()


    score = evaluate()
    if score == 1:
        import math


# Constants
PLAYER_X = 'X'
```

```
PLAYER_O = 'O'

EMPTY = ' '



# Game board

board = [[EMPTY for _ in range(3)] for _ in range(3)]



# Functions to print the board

def print_board():

  for row in board:

      print("|".join(row))

      print("-" * 5)
```

```
# Check if the game is over

def is_game_over():

  # Check rows, columns, and diagonals for a win

  for row in range(3):

      if board[row][0] == board[row][1] == board[row][2] != EMPTY:

          return True

  for col in range(3):

      if board[0][col] == board[1][col] == board[2][col] != EMPTY:

          return True

  if board[0][0] == board[1][1] == board[2][2] != EMPTY:
```

```
        return True
    if board[0][2] == board[1][1] == board[2][0] != EMPTY:
        return True
    # Check if there are any empty spaces
    for row in board:
        for cell in row:
            if cell == EMPTY:
                return False
    return True # Draw if no empty spaces


# Evaluate the board to return a score
```

```
def evaluate():
  for row in range(3):
      if board[row][0] == board[row][1] == board[row][2] != EMPTY:
          return 1 if board[row][0] == PLAYER_O else -1
  for col in range(3):
      if board[0][col] == board[1][col] == board[2][col] != EMPTY:
          return 1 if board[0][col] == PLAYER_O else -1
  if board[0][0] == board[1][1] == board[2][2] != EMPTY:
      return 1 if board[0][0] == PLAYER_O else -1
  if board[0][2] == board[1][1] == board[2][0] != EMPTY:
      return 1 if board[0][2] == PLAYER_O else -1
```

```
  return 0 # Draw



# Minimax algorithm with Alpha-Beta Pruning

def minimax(depth, is_maximizing, alpha, beta):

  score = evaluate()



  # If the game is over, return the score

  if score == 1 or score == -1:

      return score



  if is_game_over():
```

```
    return 0 # Draw



if is_maximizing:
    max_eval = -math.inf
    for row in range(3):
        for col in range(3):
            if board[row][col] == EMPTY:
                board[row][col] = PLAYER_O
                eval = minimax(depth + 1, False, alpha, beta)
                board[row][col] = EMPTY
                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] == EMPTY:
                board[row][col] = PLAYER_X
                eval = minimax(depth + 1, True, alpha, beta)
```

```
                board[row][col] = EMPTY
                min_eval = min(min_eval, eval)
                beta = min(beta, eval)
                if beta <= alpha:
                    break
        return min_eval

# Find the best move for AI
def best_move():
    best_val = -math.inf
    move = (-1, -1)
```

```
for row in range(3):
    for col in range(3):
        if board[row][col] == EMPTY:
            board[row][col] = PLAYER_O
            move_val = minimax(0, False, -math.inf, math.inf)
            board[row][col] = EMPTY
            if move_val > best_val:
                best_val = move_val
                move = (row, col)
```

```
  return move



# Get human player's move

def human_move():

  while True:

      try:

          row, col = map(int, input("Enter your move (row, col) between 0-2: ").split())

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

              board[row][col] = PLAYER_X

              break

          else:
```

```
            print("Cell is already occupied. Try again.")
        except (ValueError, IndexError):
            print("Invalid move. Please enter row and col between 0 and 2.")


# Main game loop
def play_game():
    print("Welcome to Tic-Tac-Toe!")
    print_board()

    while not is_game_over():
        # Human move
```

```
human_move()

print_board()



if is_game_over():

    break



# AI move

print("AI's move:")

ai_move = best_move()

board[ai_move[0]][ai_move[1]] = PLAYER_O

print_board()
```

```
    score = evaluate()
    if score == 1:
        print("AI wins!")
    elif score == -1:
        print("You win!")
    else:
        print("It's a draw!")

# Start the game
play_game()
```

```
print("AI wins!")

  elif score == -1:

      print("You win!")

  else:

      print("It's a draw!")



# Start the game

play_game()
```