In [1]:
```python
import numpy as np


def initialize_board():
    return np.full((3, 3), ' ')


def display_board(board):
    print("\n")
    for row in board:
        print("|".join(row))
        print("-" * 5)
    print("\n")


def check_winner(board):

    for i in range(3):
        if board[i][0] == board[i][1] == board[i][2] != ' ':
            return board[i][0]
        if board[0][i] == board[1][i] == board[2][i] != ' ':
            return board[0][i]
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return board[0][2]
    return None


def is_draw(board):
    return ' ' not in board


def make_move(board, row, col, player):
    board[row][col] = player
```

In [2]:

```
def minimax(board, depth, is_maximizing, alpha=-np.inf, beta=np.inf):
    winner = check_winner(board)
    if winner == 'X':
        return -1
    if winner == 'O':
        return 1
    if is_draw(board):
        return 0

    if is_maximizing:
        max_eval = -np.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'O'
                    eval = minimax(board, depth + 1, False, alpha, beta)
                    board[i][j] = ' '
                    max_eval = max(max_eval, eval)
                    alpha = max(alpha, eval)
                    if beta <= alpha:
                        break
        return max_eval
    else:
        min_eval = np.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'X'
                    eval = minimax(board, depth + 1, True, alpha, beta)
                    board[i][j] = ' '
                    min_eval = min(min_eval, eval)
                    beta = min(beta, eval)
                    if beta <= alpha:
                        break
        return min_eval
```

In [3]:

```
def find_best_move(board):
    best_move = None
    best_value = -np.inf
    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                board[i][j] = 'O'
                move_value = minimax(board, 0, False)
                board[i][j] = ' '
                if move_value > best_value:
                    best_value = move_value
                    best_move = (i, j)
    return best_move
```

In [*]:
```python
import numpy as np


def initialize_board():
    return np.full((3, 3), ' ')


def display_board(board):
    print("\n")
    for row in board:
        print("|".join(row))
        print("-" * 5)
    print("\n")


def check_winner(board):

    for i in range(3):
        if board[i][0] == board[i][1] == board[i][2] != ' ':
            return board[i][0]
        if board[0][i] == board[1][i] == board[2][i] != ' ':
            return board[0][i]
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return board[0][2]
    return None


def is_draw(board):
    return ' ' not in board


def make_move(board, row, col, player):
    board[row][col] = player


def minimax(board, depth, is_maximizing, alpha=-np.inf, beta=np.inf):
    winner = check_winner(board)
    if winner == 'X':
        return -1
    if winner == 'O':
        return 1
    if is_draw(board):
        return 0

    if is_maximizing:
        max_eval = -np.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'O'
                    eval = minimax(board, depth + 1, False, alpha, beta)
                    board[i][j] = ' '
                    max_eval = max(max_eval, eval)
                    alpha = max(alpha, eval)
                    if beta <= alpha:
                        break
        return max_eval
    else:
        min_eval = np.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'X'
                    eval = minimax(board, depth + 1, True, alpha, beta)
                    board[i][j] = ' '
                    min_eval = min(min_eval, eval)
                    beta = min(beta, eval)
                    if beta <= alpha:
                        break
        return min_eval

def find_best_move(board):
    best_move = None
    best_value = -np.inf
    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                board[i][j] = 'O'
                move_value = minimax(board, 0, False)
                board[i][j] = ' '
                if move_value > best_value:
                    best_value = move_value
                    best_move = (i, j)
    return best_move


def play_game():
    board = initialize_board()
    current_player = 'X'

    while True:
        display_board(board)

        if current_player == 'X':
            while True:
                try:
                    row, col = map(int, input("Enter row and column (0-2):
                    if board[row][col] != ' ':
                        print("Invalid move! The cell is already occupied.
                        continue
                    break
                except (ValueError, IndexError):
                    print("Invalid input! Please enter two numbers between
                    continue
            make_move(board, row, col, current_player)
        else:
            print("AI is making a move...")
            row, col = find_best_move(board)
            make_move(board, row, col, current_player)

        winner = check_winner(board)
        if winner:
            display_board(board)
            print(f"Player {winner} wins!")
            break
        if is_draw(board):
            display_board(board)
            print("It's a draw!")
            break
        current_player = 'O' if current_player == 'X' else 'X'


play_game()
```

```
 | |
-----
 | |
-----
 | |
-----


Enter row and column (0-2): 1 1


 | |
-----
 |X|
-----
 | |
-----


AI is making a move...


O| |
-----
 |X|
-----
 | |
-----


Enter row and column (0-2): 0 0
Invalid move! The cell is already occupied. Try again.
Enter row and column (0-2): 2 0


O| |
-----
 |X|
-----
X| |
-----


AI is making a move...


O| |O
-----
 |X|
-----
X| |
-----


Enter row and column (0-2): 0 2
Invalid move! The cell is already occupied. Try again.
Enter row and column (0-2): 0 1
```

```
O|X|O
-----
 |X|
-----
X| |
-----


AI is making a move...


O|X|O
-----
 |X|
-----
X|O|
-----


Enter row and column (0-2): 1 1
Invalid move! The cell is already occupied. Try again.
Enter row and column (0-2): 0 1
Invalid move! The cell is already occupied. Try again.

Enter row and column (0-2):
```

```
In [ ]:
```

```
In [ ]:
```