In [18]:
```
import numpy as np
import random
```

In [19]:
```
board = np.zeros((3, 3), dtype=int)
```

In [ ]:
```
def display_board(board):
    for row in board:
        print(" | ".join(["X" if cell == 1 else "O" if cell == -1 else " " for cell
        print("-" * 9)

def is_valid_move(board, row, col):
    return board[row][col] == 0

def check_win(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in board.T:
        if all(cell == player for cell in col):
            return True
    if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == playe
        return True
    return False

def check_draw(board):
    return np.all(board != 0)
```

In [20]:
```
class QLearningAgent:
    def __init__(self, epsilon, alpha, gamma):
        self.epsilon = epsilon  # Exploration rate
        self.alpha = alpha  # Learning rate
        self.gamma = gamma  # Discount factor
        self.q_table = {}  # Q-value table

    def get_action(self, state):
        state_tuple = tuple(map(tuple, state))
        if np.random.rand() < self.epsilon:
            valid_moves = [i for i in range(9) if state[i // 3][i % 3] == 0]
            return random.choice(valid_moves)
        else:
            return max(
                (i for i in range(9) if state[i // 3][i % 3] == 0),
                key=lambda i: self.q_table.get(state_tuple, {}).get(i, 0),
                default=random.choice([i for i in range(9) if state[i // 3][i % 3]
            )

    def learn(self, state, action, reward, next_state):
        state_tuple = tuple(map(tuple, state))
        next_state_tuple = tuple(map(tuple, next_state))
        if state_tuple not in self.q_table:
            self.q_table[state_tuple] = {}
        if next_state_tuple not in self.q_table:
            self.q_table[next_state_tuple] = {}
```

```
        if action not in self.q_table[state_tuple]:
            self.q_table[state_tuple][action] = 0

        best_next_action = max(
            (i for i in range(9) if next_state[i // 3][i % 3] == 0),
            key=lambda i: self.q_table.get(next_state_tuple, {}).get(i, 0),
            default=None
        )

        if best_next_action is not None:
            self.q_table[state_tuple][action] += self.alpha * (
                reward + self.gamma * self.q_table.get(next_state_tuple, {}).get(be
                - self.q_table[state_tuple].get(action, 0)
            )
        else:
            self.q_table[state_tuple][action] += self.alpha * (reward - self.q_tabl
```

In [21]:

```
def play_game(agent1, agent2, board):
    state = board.copy()
    while True:
        action1 = agent1.get_action(state)
        row, col = divmod(action1, 3)
        state[row][col] = 1

        if check_win(state, 1):
            agent1.learn(state, action1, 1, state)
            return 1  # Agent 1 wins

        if check_draw(state):
            return 0  # Draw

        action2 = agent2.get_action(state)
        row, col = divmod(action2, 3)
        state[row][col] = -1

        if check_win(state, -1):
            agent1.learn(state, action1, -1, state)
            return -1  # Agent 2 wins

def train_q_learning_agents(agent1, agent2, num_episodes):
    for episode in range(num_episodes):
        board = np.zeros((3, 3), dtype=int)
        if episode % 2 == 0:
            result = play_game(agent1, agent2, board)
            if result == 1:
                agent1.learn(board, None, 1, board)
                agent2.learn(board, None, -1, board)
            elif result == -1:
                agent1.learn(board, None, -1, board)
                agent2.learn(board, None, 1, board)
            else:
                agent1.learn(board, None, 0, board)
                agent2.learn(board, None, 0, board)
        else:
            result = play_game(agent2, agent1, board)
            if result == 1:
```

```
                agent1.learn(board, None, -1, board)
                agent2.learn(board, None, 1, board)
            elif result == -1:
                agent1.learn(board, None, 1, board)
                agent2.learn(board, None, -1, board)
            else:
                agent1.learn(board, None, 0, board)
                agent2.learn(board, None, 0, board)

agent1 = QLearningAgent(epsilon=0.2, alpha=0.1, gamma=0.9)
agent2 = QLearningAgent(epsilon=0.2, alpha=0.1, gamma=0.9)

train_q_learning_agents(agent1, agent2, num_episodes=10000)

board = np.zeros((3, 3), dtype=int)
```

In [22]:

```
while True:
    display_board(board)
    action1 = agent1.get_action(board)
    row, col = divmod(action1, 3)
    board[row][col] = 1
    if check_win(board, 1):
        display_board(board)
        print("Agent 1 wins!")
        break
    if check_draw(board):
        display_board(board)
        print("It's a draw!")
        break
    display_board(board)
    action2 = agent2.get_action(board)
    row, col = divmod(action2, 3)
    board[row][col] = -1
    if check_win(board, -1):
        display_board(board)
        print("Agent 2 wins!")
        break
```

```
  |   |
---------
  |   |
---------
  |   |
---------
  | X |
---------
  |   |
---------
  |   |
---------
O | X |
---------
  |   |
---------
  |   |
---------
O | X | X
---------
  |   |
---------
  |   |
---------
O | X | X
---------
O |   |
---------
  |   |
---------
O | X | X
---------
O | X |
---------
  |   |
---------
O | X | X
---------
O | X | O
---------
  |   |
---------
O | X | X
---------
O | X | O
---------
X |   |
---------
Agent 1 wins!
```