In [2]:
```
'''
NAME: MANE SHIVRAJ PANDURANG
ROLL NO.37
COURSE: AI&DS, SUB:ML(Machine Learning)
CLASS: BE
'''
```

Out[2]:
```
'\nNAME: MANE SHIVRAJ PANDURANG\nROLL NO.37\nCOURSE: AI&DS, SUB:ML(Machine Learning)\nCLASS: BE \n'
```

In [3]:
```
'''
    PRACTICAL.NO:06
    Build a Tic-Tac-Toe game using reinforcement learning in Python by using following tasks
        a. Setting up the environment
        b. Defining the Tic-Tac-Toe game
        c. Building the reinforcement learning model
        d. Training the model
        e. Testing the model
'''
```

Out[3]:
```
'\n    PRACTICAL.NO:06\n    Build a Tic-Tac-Toe game using reinforcement learning in Python by using following tasks
\n        a. Setting up the environment\n        b. Defining the Tic-Tac-Toe game\n        c. Building the reinforce
ment learning model\n        d. Training the model\n        e. Testing the model\n'
```

In [4]:
```
import numpy as np
```

In [5]:
```
class TicTacToeEnvironment:
  def __init__(self):
   self.state = [0] * 9
   self.is_terminal = False
  def reset(self):
   self.state = [0] * 9
   self.is_terminal = False
  def get_available_moves(self):
   return [i for i, mark in enumerate(self.state) if mark == 0]

  def make_move(self, move, player_mark):
   self.state[move] = player_mark

  def check_win(self, player_mark):
    winning_states = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8], # rows
        [0, 3, 6], [1, 4, 7], [2, 5, 8], # columns
        [0, 4, 8], [2, 4, 6] # diagonals
        ]
    for state_indices in winning_states:
      if all(self.state[i] == player_mark for i in state_indices):
        self.is_terminal = True
        return True
    return False

  def is_draw(self):
    return 0 not in self.state
```

In [6]:
```
class QLearningAgent:
  def __init__(self, learning_rate=0.9, discount_factor=0.9, exploration_rate=0.3):
    self.learning_rate = learning_rate
    self.discount_factor = discount_factor
    self.exploration_rate = exploration_rate
    self.q_table = np.zeros((3**9, 9))

  def get_state_index(self, state):
    state_index = 0

    for i, mark in enumerate(state):
      state_index += (3 ** i) * (mark + 1)
      return state_index

  def choose_action(self, state, available_moves):
    state_index = self.get_state_index(state)

    if np.random.random() < self.exploration_rate:
      return np.random.choice(available_moves)
    else:
      return np.argmax(self.q_table[state_index, available_moves])

  def update_q_table(self, state, action, next_state, reward):
    state_index = self.get_state_index(state)
    next_state_index = self.get_state_index(next_state) if next_state is not None else None
    max_q_value = np.max(self.q_table[next_state_index]) if next_state is not None else 0
    self.q_table[state_index, action] = (1 - self.learning_rate) * self.q_table[state_index, action] + \
    self.learning_rate * (reward + self.discount_factor * max_q_value)
```

In [7]:

```
def evaluate_agents(agent1, agent2, num_episodes=1000):
  environment = TicTacToeEnvironment()
  agent1_wins = 0
  agent2_wins = 0
  draws = 0

  for _ in range(num_episodes):
    environment.reset()
    current_agent = agent1

    while not environment.is_terminal:
      available_moves = environment.get_available_moves()

      current_state = environment.state.copy()
      action = current_agent.choose_action(current_state, available_moves)
      environment.make_move(action, 1 if current_agent == agent1 else -1)

      if environment.check_win(1 if current_agent == agent1 else -1):
        current_agent.update_q_table(current_state, action, None, 10)

        if current_agent == agent1:
          agent1_wins += 1
        else:
          agent2_wins += 1
          break
      elif environment.is_draw():
        current_agent.update_q_table(current_state, action, None, 0)
        draws += 1
        break
      next_state = environment.state.copy()
      reward = 0

      if environment.check_win(1 if current_agent == agent1 else -1):
        reward = -10

        current_agent.update_q_table(current_state, action, next_state, reward)
        current_agent = agent2 if current_agent == agent1 else agent1
  return agent1_wins, agent2_wins, draws
```

In [8]:

```
# Create agents
agent1 = QLearningAgent()
agent2 = QLearningAgent()
# Evaluate agents
agent1_wins, agent2_wins, draws = evaluate_agents(agent1, agent2)
# Print results
print(f"Agent 1 wins: {agent1_wins}")
print(f"Agent 2 wins: {agent2_wins}")
print(f"Draws: {draws}")
# Agent 1 wins: 458
# # Agent 2 wins: 470
# Draws: 72
```

```
Agent 1 wins: 1000
Agent 2 wins: 0
Draws: 0
```