**Practical – 9**

Aim – Implement basic version of chess game using Reinforcement Learning. **Code**

import numpy as np

import random

class ChessEnvironment: def \_\_init\_\_(self):

self.board = self.initialize\_board() self.state\_size = 64 # 8x8 board

self.action\_size = 4096 # 64\*64 possible moves

def initialize\_board(self):

board = np.zeros((8, 8), dtype=int) board[0] = [-1, -2, -3, -4, -5, -3, -2, -1] board[1] = [-6] \* 8

board[6] = [6] \* 8

board[7] = [1, 2, 3, 4, 5, 3, 2, 1] return board

def get\_valid\_moves(self):

# Return a list of valid moves in the current state valid\_moves = []

# Implement logic to generate valid moves here return valid\_moves

def get\_state(self):

# Flatten the board to create the state representation return self.board.flatten()

def step(self, action):

# Apply the action to the environment and return the next state, reward, and whether the game is done

next\_state = self.get\_state() # Placeholder reward = 0 # Placeholder

done = False # Placeholder

return next\_state, reward, done

class QLearningAgent:

def \_\_init\_\_(self, state\_size, action\_size):

self.state\_size = state\_size self.action\_size = action\_size

self.q\_table = np.zeros((state\_size, action\_size)) self.learning\_rate = 0.1

self.discount\_factor = 0.99

self.epsilon = 1.0

self.epsilon\_decay = 0.999

self.epsilon\_min = 0.01

def choose\_action(self, state):

if np.random.rand() <= self.epsilon:

return random.randrange(self.action\_size) else:

return np.argmax(self.q\_table[state, :])

def learn(self, state, action, reward, next\_state, done):

target = reward + self.discount\_factor \* np.max(self.q\_table[next\_state, :])

self.q\_table[state, action] += self.learning\_rate \* (target - self.q\_table[state, action])

if done:

self.epsilon = max(self.epsilon\_min, self.epsilon \* self.epsilon\_decay)

def main():

env = ChessEnvironment()

agent = QLearningAgent(env.state\_size, env.action\_size)

episodes = 1000 for episode in range(episodes):

state = env.get\_state() done = False total\_reward = 0

while not done:

action = agent.choose\_action(state) next\_state, reward, done = env.step(action) agent.learn(state, action, reward, next\_state, done) total\_reward += reward

state = next\_state

print(f"Episode: {episode + 1}, Total Reward: {total\_reward}")

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

main()

**Output**

