**Assignment 4**

**Experiment 10**

Title: Q-Learning and Backpropagation

Name of Student: Sangeet Agrawal PRN No. 21070122140

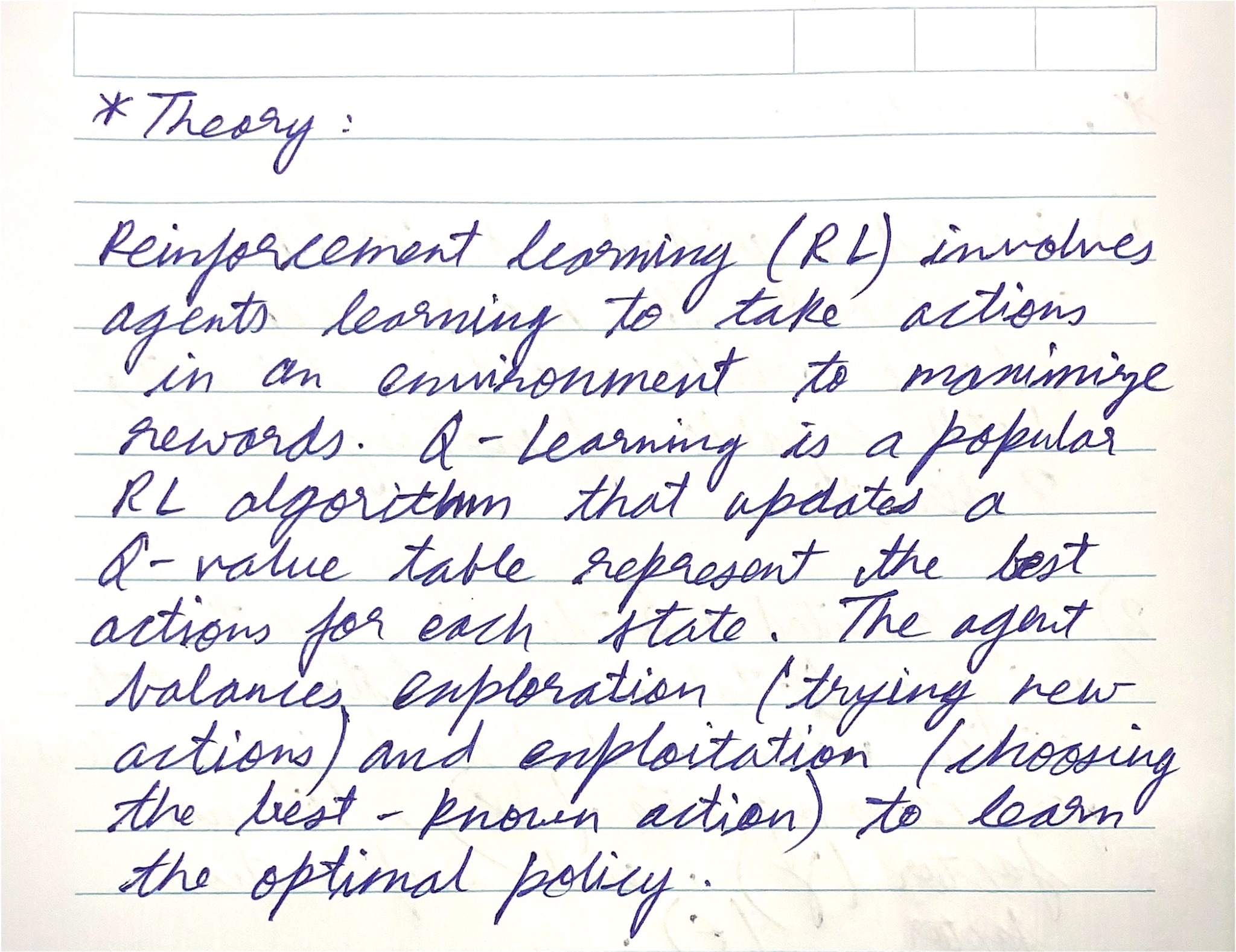
DoP: 7 Oct DoS: 9 Oct

**Aim:** Study and Implementation of  Q-learning

**Problem Statement:** Develop a game using Reinforcement learning (Q learning).

Rules are as follows:

* Travelers have 8 degrees of movement. Up, down, left, and right.
* Create a maze as discussed in the class.



Problem Statement 1 :

To demonstrate Reinforcement Learning at work, You can develop an applet that uses RL methods and learns to play a game. The particular game was Cat and Mouse. For those of you unfamiliar with the game, it is a simple game. There is a cat, a mouse, a piece of cheese as well as some obstacles in the cat and mouse world. The mouse tries to avoid getting caught by the cat, at the same time trying to get to the cheese to eat it. The mouse is the one learning in the applet, the cat is already programmed to go for the mouse. When the applet loads, either choose one of the pre-specified obstacle layouts or have one randomly generated for you.

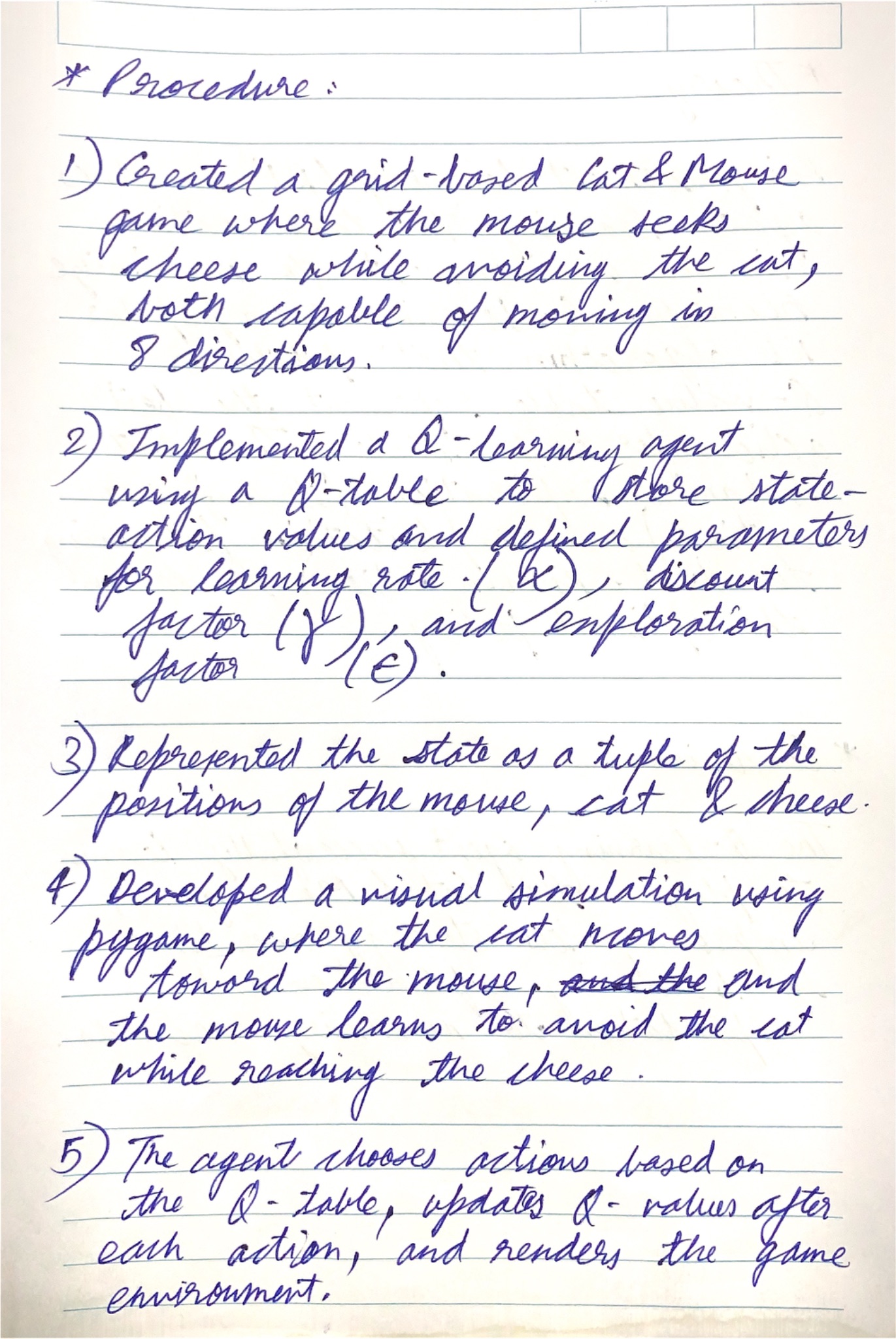
You can use any other platform for the same.

The rules of the Cat and Mouse game are:

* Both the cat and mouse have 8 degrees of movement. Up, down, left and right, as well as the four diagonals.
* The mouse scores a point for getting the cheese. The mouse gets the cheese when it is in the same square as the cheese.
* The cat scores a point for catching the mouse, by simply moving to the same square as the mouse.
* If the mouse gets the cheese, a new piece is placed randomly while the cat and mouse keep their positions.
* The game is over when the cat catches the mouse. The scores are then updated and a new game can begin.

Hardware/Software:

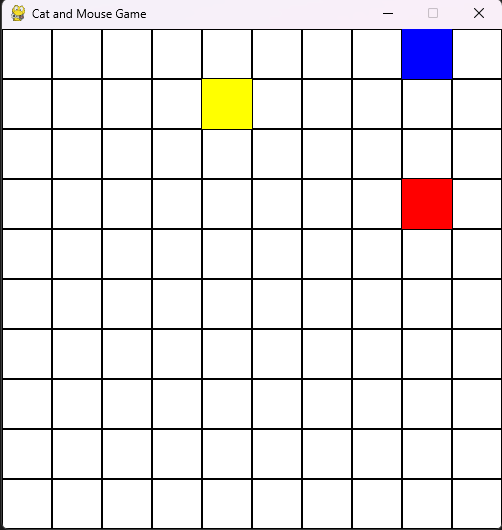
* Processors. Minimum: Any Intel or AMD x86-64 processor. Recommended: Any Intel or AMD x86-64 processor with four logical cores and AVX2 instruction set support.
* Disk. Minimum: 3.4 GB of disk space for MATLAB only, 5-8 GB for a typical installation.
* Recommended: An SSD is recommended
* RAM. Minimum: 4 GB. Recommended: 8 GB.
* Software: MATLAB R2023a.

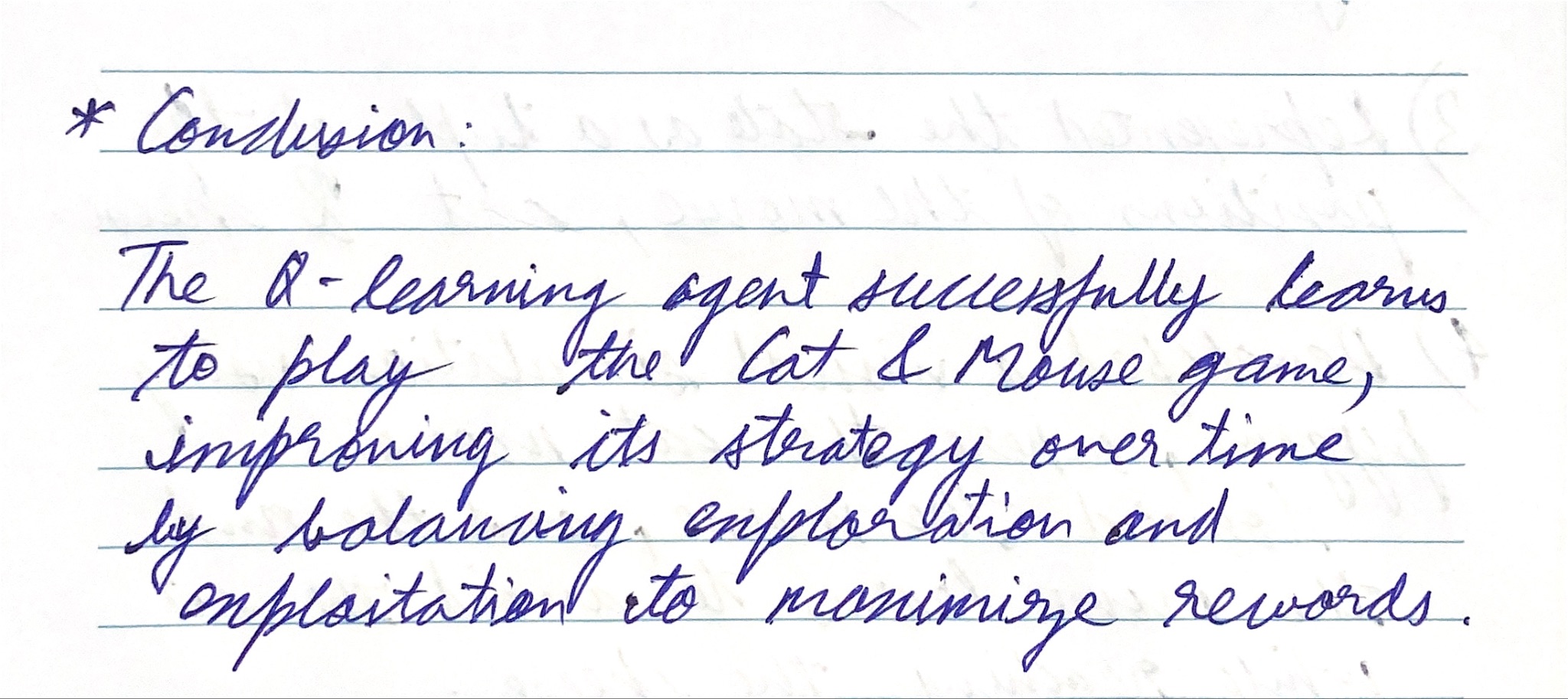


**Code:**

| import numpy as np import random import pygame  # Constants GRID\_SIZE = 10 ACTIONS = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (-1, -1), (-1, 1), (1, -1)] # 8 degrees of movement ALPHA = 0.1 # Learning rate GAMMA = 0.9 # Discount factor EPSILON = 0.1 # Exploration factor CELL\_SIZE = 50 # Size of each cell in the grid  # Initialize pygame pygame.init() screen = pygame.display.set\_mode((GRID\_SIZE \* CELL\_SIZE, GRID\_SIZE \* CELL\_SIZE)) pygame.display.set\_caption("Cat and Mouse Game") clock = pygame.time.Clock()  # Define colors WHITE = (255, 255, 255) BLACK = (0, 0, 0) RED = (255, 0, 0) BLUE = (0, 0, 255) YELLOW = (255, 255, 0)  # Define the environment (Cat and Mouse Grid) class CatMouseEnv:  def \_\_init\_\_(self, grid\_size):  self.grid\_size = grid\_size  self.reset()   def reset(self):  # Place mouse, cat, and cheese in random positions  self.mouse\_pos = [random.randint(0, self.grid\_size - 1), random.randint(0, self.grid\_size - 1)]  self.cat\_pos = [random.randint(0, self.grid\_size - 1), random.randint(0, self.grid\_size - 1)]  self.cheese\_pos = [random.randint(0, self.grid\_size - 1), random.randint(0, self.grid\_size - 1)]  return self.get\_state()   def get\_state(self):  return tuple(self.mouse\_pos + self.cat\_pos + self.cheese\_pos)   def is\_valid(self, pos):  return 0 <= pos[0] < self.grid\_size and 0 <= pos[1] < self.grid\_size   def step(self, action):  # Move mouse  new\_mouse\_pos = [self.mouse\_pos[0] + action[0], self.mouse\_pos[1] + action[1]]  if self.is\_valid(new\_mouse\_pos):  self.mouse\_pos = new\_mouse\_pos   # Move cat towards mouse  self.move\_cat()   # Check for win/lose condition  if self.mouse\_pos == self.cheese\_pos:  reward = 1 # Mouse gets cheese  done = False # Game continues after getting the cheese  self.cheese\_pos = [random.randint(0, self.grid\_size - 1), random.randint(0, self.grid\_size - 1)] # New cheese position  elif self.mouse\_pos == self.cat\_pos:  reward = -1 # Cat catches mouse  done = True # Game ends when the cat catches the mouse  else:  reward = 0 # No one has won yet  done = False   return self.get\_state(), reward, done   def move\_cat(self):  if self.cat\_pos[0] < self.mouse\_pos[0]:  self.cat\_pos[0] += 1  elif self.cat\_pos[0] > self.mouse\_pos[0]:  self.cat\_pos[0] -= 1   if self.cat\_pos[1] < self.mouse\_pos[1]:  self.cat\_pos[1] += 1  elif self.cat\_pos[1] > self.mouse\_pos[1]:  self.cat\_pos[1] -= 1   def render(self):  # Draw grid  screen.fill(WHITE)  for row in range(self.grid\_size):  for col in range(self.grid\_size):  pygame.draw.rect(screen, BLACK, pygame.Rect(col \* CELL\_SIZE, row \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE), 1)   # Draw mouse, cat, and cheese  pygame.draw.rect(screen, BLUE, pygame.Rect(self.mouse\_pos[1] \* CELL\_SIZE, self.mouse\_pos[0] \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))  pygame.draw.rect(screen, RED, pygame.Rect(self.cat\_pos[1] \* CELL\_SIZE, self.cat\_pos[0] \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))  pygame.draw.rect(screen, YELLOW, pygame.Rect(self.cheese\_pos[1] \* CELL\_SIZE, self.cheese\_pos[0] \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))  pygame.display.flip()  # Q-learning Agent class QLearningAgent:  def \_\_init\_\_(self, actions, alpha=ALPHA, gamma=GAMMA, epsilon=EPSILON):  self.q\_table = {} # Q-table: maps state-action pairs to rewards  self.actions = actions  self.alpha = alpha  self.gamma = gamma  self.epsilon = epsilon   def get\_q\_value(self, state, action):  return self.q\_table.get((state, action), 0.0)   def update\_q\_value(self, state, action, reward, next\_state):  old\_q\_value = self.get\_q\_value(state, action)  max\_next\_q\_value = max([self.get\_q\_value(next\_state, a) for a in self.actions])  new\_q\_value = old\_q\_value + self.alpha \* (reward + self.gamma \* max\_next\_q\_value - old\_q\_value)  self.q\_table[(state, action)] = new\_q\_value   def choose\_action(self, state):  if random.uniform(0, 1) < self.epsilon: # Explore  return random.choice(self.actions)  else: # Exploit  q\_values = [self.get\_q\_value(state, action) for action in self.actions]  max\_q = max(q\_values)  return self.actions[q\_values.index(max\_q)]  # Main loop to run the simulation with pygame visualization def run\_simulation(episodes=500):  env = CatMouseEnv(GRID\_SIZE)  agent = QLearningAgent(ACTIONS)   for episode in range(episodes):  state = env.reset()  done = False  while not done:  # Handle Pygame events (for closing the window)  for event in pygame.event.get():  if event.type == pygame.QUIT:  pygame.quit()  return   # Choose action and take a step in the environment  action = agent.choose\_action(state)  next\_state, reward, done = env.step(action)  agent.update\_q\_value(state, action, reward, next\_state)  state = next\_state   # Render the environment  env.render()   # Control the frame rate  clock.tick(5)   if done:  break # Terminate the game if the cat catches the mouse   if episode % 50 == 0:  print(f"Episode {episode}: Q-table size {len(agent.q\_table)}")   if \_\_name\_\_ == "\_\_main\_\_":  run\_simulation() |
| --- |

**Output:**



****