Instituto Tecnológico y de Estudios Superiores de Monterrey

ESCUELA DE INGENIERÍA Y CIENCIAS

Inteligencia Artificial Avanzada para la Ciencia de Datos II

A4. Labyrinths with Q-Learning

Presenta:

Miguel Ángel Pérez Ávila - A01369908

Profesor:

Dr. Gerardo Jesús Camacho González

Sante Fe, Ciudad de México a 26 de Octubre del 2025

Resultados

Para la ejecución del programa se utilizaron los siguientes hiper parámetros:

```
    # Tamaño del grid para el entorno
    SIZE = 10
    # Número de acciones posibles
    ACTIONS = 3
    # Número de episodios para el entrenamiento
    EPISODES = 5000
    # Número de pasos por episodio
    STEPS = 400
    # Factor para el descuento y propagación del Q-value en la tabla
    DISCOUNT_FACTOR = 0.98
    # Learning rate
    LR = 0.03
```

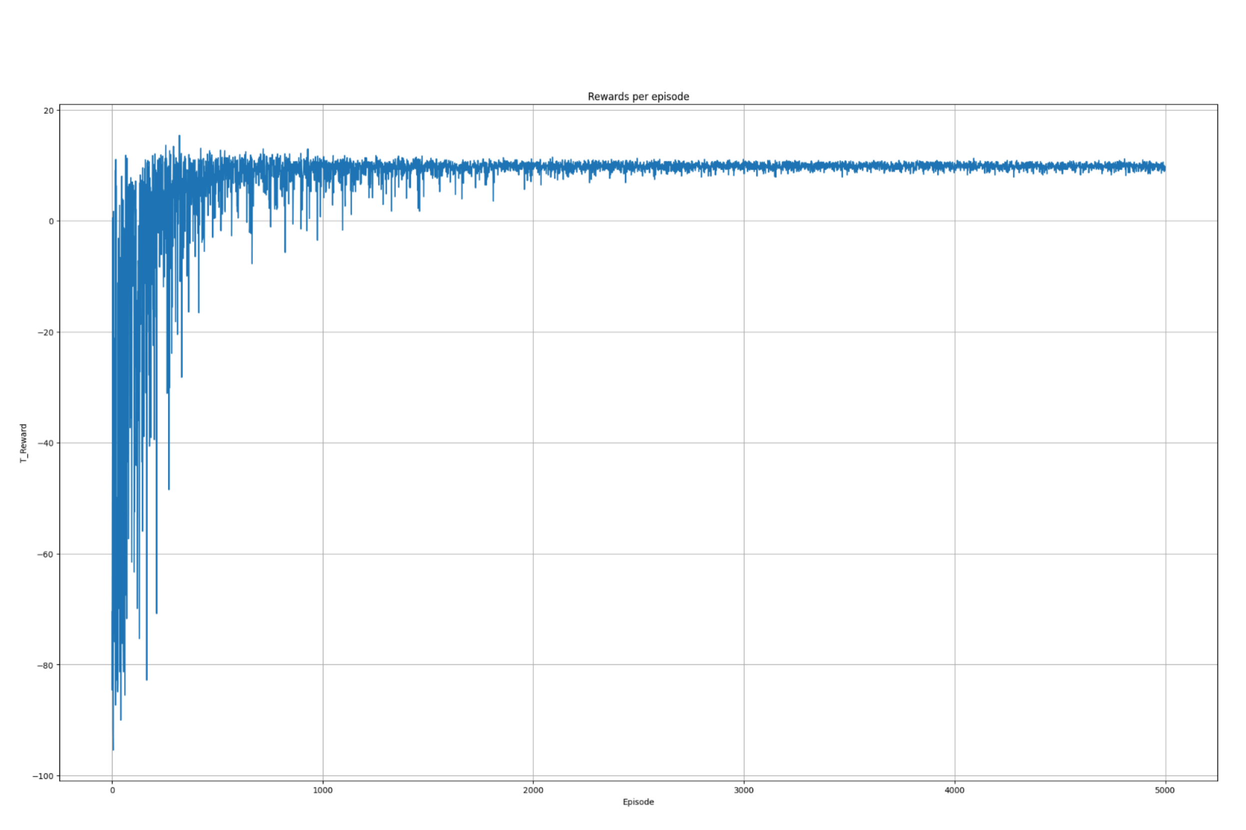
Y se obtuvieron los siguientes resultados del proceso de entrenamiento:

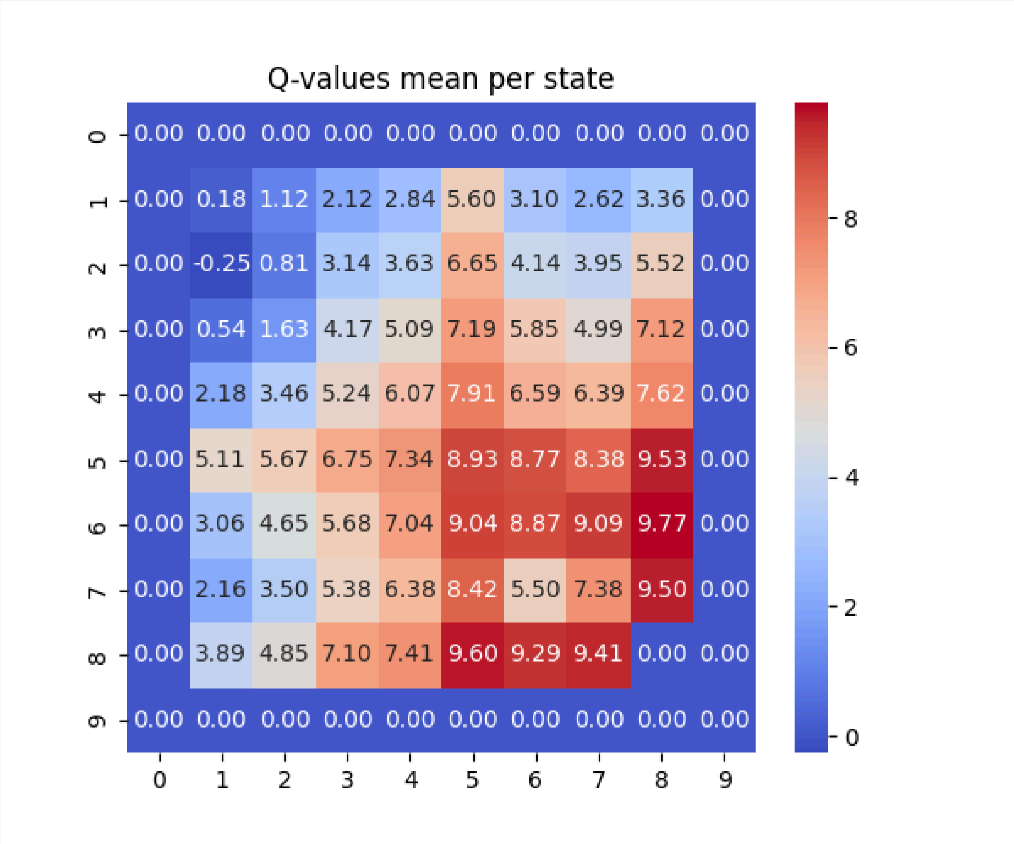
```
Entrenamiento del agente...

Episode 100/5000 -- ε=0.910 -- Reward=-11.77
Episode 200/5000 -- ε=0.828 -- Reward=-2.97
Episode 300/5000 -- ε=0.754 -- Reward=10.17
Episode 400/5000 -- ε=0.687 -- Reward=3.90
Episode 500/5000 -- ε=0.626 -- Reward=10.69
Episode 600/5000 -- ε=0.571 -- Reward=0.52
Episode 700/5000 -- ε=0.522 -- Reward=11.48
Episode 800/5000 -- ε=0.477 -- Reward=11.78
Episode 900/5000 -- ε=0.436 -- Reward=10.98
Episode 1000/5000 -- ε=0.399 -- Reward=9.68
Episode 1100/5000 -- ε=0.366 -- Reward=9.88
Episode 1200/5000 -- ε=0.336 -- Reward=8.68
Episode 1300/5000 -- ε=0.309 -- Reward=10.39
Episode 1400/5000 -- ε=0.284 -- Reward=9.09
Episode 1500/5000 -- ε=0.262 -- Reward=9.29
Episode 1600/5000 -- ε=0.242 -- Reward=10.88
Episode 1700/5000 -- ε=0.224 -- Reward=9.49
Episode 1800/5000 -- ε=0.207 -- Reward=9.28
Episode 1900/5000 -- ε=0.192 -- Reward=9.49
Episode 2000/5000 -- ε=0.179 -- Reward=9.89
Episode 2100/5000 -- ε=0.166 -- Reward=10.28
Episode 2200/5000 -- ε=0.155 -- Reward=10.59
Episode 2300/5000 -- ε=0.145 -- Reward=9.19
```

```
Episode 2400/5000 -- ε=0.136 -- Reward=9.28
Episode 2500/5000 -- ε=0.128 -- Reward=9.69
Episode 2600/5000 -- ε=0.121 -- Reward=9.89
Episode 2700/5000 -- ε=0.114 -- Reward=8.88
Episode 2800/5000 -- ε=0.108 -- Reward=9.49
Episode 2900/5000 -- ε=0.102 -- Reward=9.49
Episode 3000/5000 -- ε=0.097 -- Reward=9.29
Episode 3100/5000 -- ε=0.093 -- Reward=9.48
Episode 3200/5000 -- ε=0.089 -- Reward=10.39
Episode 3300/5000 -- ε=0.085 -- Reward=9.89
Episode 3400/5000 -- ε=0.082 -- Reward=9.49
Episode 3500/5000 -- ε=0.079 -- Reward=10.69
Episode 3600/5000 -- ε=0.076 -- Reward=10.29
Episode 3700/5000 -- ε=0.073 -- Reward=10.69
Episode 3800/5000 -- ε=0.071 -- Reward=9.99
Episode 3900/5000 -- ε=0.069 -- Reward=10.49
Episode 4000/5000 -- ε=0.067 -- Reward=9.89
Episode 4100/5000 -- ε=0.066 -- Reward=9.29
Episode 4200/5000 -- ε=0.064 -- Reward=10.29
Episode 4300/5000 -- ε=0.063 -- Reward=9.79
Episode 4400/5000 -- ε=0.062 -- Reward=9.69
Episode 4500/5000 -- ε=0.061 -- Reward=9.08
Episode 4600/5000 -- ε=0.060 -- Reward=9.68
Episode 4700/5000 -- ε=0.059 -- Reward=9.29
Episode 4800/5000 -- ε=0.058 -- Reward=10.49
Episode 4900/5000 -- ε=0.057 -- Reward=10.49
Episode 5000/5000 -- ε=0.056 -- Reward=9.79

Success rate: 4954/5000
```


Rewards per episode
T_Reward
Episode
20
0
−20
−40
−60
−80
−100
0
1000
2000
3000
4000
5000


Q-values mean per state
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.18 1.12 2.12 2.84 5.60 3.10 2.62 3.36 0.00
0.00 -0.25 0.81 3.14 3.63 6.65 4.14 3.95 5.52 0.00
0.00 0.54 1.63 4.17 5.09 7.19 5.85 4.99 7.12 0.00
0.00 2.18 3.46 5.24 6.07 7.91 6.59 6.39 7.62 0.00
0.00 5.11 5.67 6.75 7.34 8.93 8.77 8.38 9.53 0.00
0.00 3.06 4.65 5.68 7.04 9.04 8.87 9.09 9.77 0.00
0.00 2.16 3.50 5.38 6.38 8.42 5.50 7.38 9.50 0.00
0.00 3.89 4.85 7.10 7.41 9.60 9.29 9.41 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0 1 2 3 4 5 6 7 8 9
8
6
4
2
0

Finalmente, en el entorno de prueba, se obtuvieron los siguiente resultados:

```
Test del agente entrenado...

- Test Episode 1 -- Success=True, Total Reward=9.70, Steps=7
- Test Episode 2 -- Success=True, Total Reward=10.20, Steps=9
- Test Episode 3 -- Success=True, Total Reward=9.70, Steps=7
- Test Episode 4 -- Success=True, Total Reward=9.80, Steps=11
- Test Episode 5 -- Success=True, Total Reward=10.60, Steps=7
- Test Episode 6 -- Success=True, Total Reward=9.80, Steps=11
- Test Episode 7 -- Success=True, Total Reward=9.30, Steps=15
- Test Episode 8 -- Success=True, Total Reward=10.50, Steps=9
- Test Episode 9 -- Success=True, Total Reward=10.10, Steps=11
- Test Episode 10 -- Success=True, Total Reward=9.90, Steps=6

Success rate during test: 10/10
```

Código

Implementación de Tabular QLearning:

```
# Importación de librerias
from minigrid.wrappers import RGBImgObsWrapper
from minigrid_simple_env import SimpleEnv
import numpy as np
import matplotlib.pyplot as plt

# Función que Inicializa la tabla Q con ceros para todos los estados y
acciones posibles
def init_Q_Table(size, n_actions):
    q_table = {}
    for i in range(size):
        for j in range(size):
            for d in range(4): # Direcciones posibles
                q_table[(i, j, d)] = np.zeros(shape=n_actions) # Vector Q
por cada acción
    return q_table

# Ecuación de actualización de Q-Learning
```

```python
# Q(s,a) ← Q(s,a) + α * [r + γ * max(Q(s',a')) - Q(s,a)]
def q_learning_eq(lr, reward, discount_factor, Qk, maxQ):
    return Qk + lr * (reward + discount_factor * maxQ - Qk)

# Devuelve el valor máximo de Q y la acción asociada para un estado dado
def Qmax_state(Qtable, current_state):
    q_values = Qtable[current_state]
    max_idx = np.argmax(q_values)
    max_val = q_values[max_idx]
    return max_val, max_idx

# Entrenamiento del agente mediante Q-Learning
def train(env, qTable, EPISODES, STEPS, ACTIONS, LR, DISCOUNT_FACTOR):
    print("\nEntrenamiento del agente...\n")

    # Parámetros de exploración
    max_epsilon = 1.0
    min_epsilon = 0.05
    decay_rate = 0.001

    # Contador de episodios exitosos
    success_count = 0

    # Registro de recompensas por episodio
    rewards_per_episode = []

    # Entrenamiento
    for episode in range(1, EPISODES + 1):
        total_reward = 0

        # Calculo de epsilon
        epsilon = min_epsilon + (max_epsilon - min_epsilon) *
np.exp(-decay_rate * episode)

        # Reiniciar el entorno
        obs, _ = env.reset()
        terminated = False

        # Iteración de los pasos definidos
        for step in range(STEPS):
            # Get current position and direction
            pos = tuple(env.unwrapped.agent_pos)
            dir = env.unwrapped.agent_dir
```

```
            current_state = (pos[0], pos[1], dir)

            # Selección de acción
            if np.random.uniform(0, 1) < epsilon:
                # Acción aleatoria (exploración)
                action = np.random.randint(ACTIONS)
            else:
                # Mejor acción según la Q-table (explotación)
                _, action = Qmax_state(qTable, current_state)

            # Ejecutar la acción en el entorno
            obs, reward, terminated, truncated, info = env.step(action)

            # Agregar ligera penalización para recompensar soluciones más
rápidas
            reward -= 0.001
            total_reward += reward

            # Obtener nuevo estado después de la acción
            new_pos = tuple(env.unwrapped.agent_pos)
            new_dir = env.unwrapped.agent_dir
            next_state = (new_pos[0], new_pos[1], new_dir)

            # Actualizar valor Q usando la ecuación de Q-Learninge
            qmax = np.max(qTable[next_state])
            qTable[current_state][action] = q_learning_eq(LR, reward,
DISCOUNT_FACTOR, qTable[current_state][action], qmax)

            # Episodio finalizado
            if terminated or truncated:
                if terminated:
                    # Si se llego a la meta, agregar a la cuenta de
episodios exitosos
                    success_count += 1
                break

        # Guardar recompensa total del episodio
        rewards_per_episode.append(total_reward)

        # Log
        if episode % 100 == 0:
            print(f"Episode {episode}/{EPISODES} -- ε={epsilon:.3f} --
Reward={total_reward:.2f}")
```

```
    # Mostrar tasa de éxito final
    print(f"\nSuccess rate: {success_count}/{EPISODES}")

    return qTable, rewards_per_episode

# Evaluación del agente entrenado sin exploración
def test(env, qTable, STEPS, SIZE, EPISODES):
    print("\nTest del agente entrenado...\n")

    # Crear entorno con renderizado visual
    env = SimpleEnv(size=SIZE, render_mode="human")
    env = RGBImgObsWrapper(env)

    success_count = 0

    # Ejecutar episodios de prueba
    for episode in range(1, EPISODES + 1):
        obs, _ = env.reset()
        terminated = False
        total_reward = 0
        steps = 0

        # Ejecutar pasos hasta que el episodio termine
        while not terminated and steps < STEPS:
            # Obtener el estado actual
            pos = tuple(env.unwrapped.agent_pos)
            dir = env.unwrapped.agent_dir
            current_state = (pos[0], pos[1], dir)

            # Seleccionar la mejor acción según la Q-table
            _, q_action = Qmax_state(qTable, current_state)

            # Ejecutar acción seleccionada
            obs, reward, terminated, truncated, info = env.step(q_action)
            total_reward += reward
            steps += 1

            if terminated or truncated:
                # Si llega al objetivo o se termina el episodio cortar el
ciclo
                break
```

```
        if terminated:
            success_count += 1

        print(f"- Test Episode {episode} -- Success={terminated}, Total
Reward={total_reward:.2f}, Steps={steps}")

    # Mostrar resultados finales
    print(f"\nSuccess rate during test: {success_count}/{EPISODES}")
```

Módulo del entorno para minigrid:

```
from __future__ import annotations
from minigrid.core.grid import Grid
from minigrid.core.mission import MissionSpace
from minigrid.core.world_object import  Goal
from minigrid.minigrid_env import MiniGridEnv
import random

class SimpleEnv(MiniGridEnv):
    def __init__(
        self,
        size=19,
        max_steps: int | None = None,
        **kwargs,
    ):
        self.size = size
        self.key_positions = []
        self.lava_positions = []

        self.start_agent_pos=(1,1)

        mission_space = MissionSpace(mission_func=self._gen_mission)

        if max_steps is None:
            max_steps = 4 * size**2

        super().__init__(
            mission_space=mission_space,
            grid_size=size,
            see_through_walls=True,
            max_steps=max_steps,
            **kwargs,
        )
```

```
    @staticmethod
    def _gen_mission():
        return "Reach the goal"

    def _gen_grid(self, width, height):
        self.grid = Grid(width, height)
        self.grid.wall_rect(0, 0, width, height)

        # Place walls in straight lines
        # Vertical walls
        ##for y in range(1, height-1):
        #    self.put_obj(Wall(), width // 2, y)

        # Horizontal walls
        #for x in range(1, width-1):
        #    self.put_obj(Wall(), x, height//2)

        # Create openings in the walls
        #openings =
[(width//2,5),(width//2,15),(5,height//2),(15,height//2),]

        #for x, y in openings:
        #    self.grid.set(x, y, None)


        # Place a goal square in the bottom-right corner
        self.goal_pos = (width - 2, height - 2)
        self.put_obj(Goal(), *self.goal_pos)

        self._place_agent()

        self.mission = "Reach the goal"

    def _place_agent(self):
        # Evitar colocar al agente cerca del objetivo
        min_distance = self.size // 2  # distancia mínima al goal

        while True:
            x = random.randint(1, self.size - 2)
            y = random.randint(1, self.size - 2)
            pos = (x, y)
```

```
        # Calcular distancia Manhattan al goal
        goal_x, goal_y = self.goal_pos
        distance = abs(goal_x - x) + abs(goal_y - y)

        # Asegurarse de que el lugar esté vacío y lejos del objetivo
        if (
            self.grid.get(*pos) is None and
            pos != self.goal_pos and
            distance >= min_distance
        ):
            self.agent_pos = pos
            self.agent_dir = random.randint(0, 3)
            break


def reset(self, **kwargs):
    #print("resetting")
    self.stepped_floors = set()
    obs = super().reset(**kwargs)
    # self._place_agent()  # Place the agent in a new random position
    return obs

def step(self, action):
    prev_pos=self.agent_pos
    prev_dir=self.agent_dir
    obs, reward, terminated, truncated, info = super().step(action)

    SIZE = self.size-2

    reward = -0.2  # base penalty

    if self.agent_pos[0] > SIZE//2 and self.agent_pos[1] > SIZE//2:
        reward += 0.3  # incentivo por acercarse al goal

    if prev_dir == self.agent_dir and prev_pos == self.agent_pos:
        reward -= 0.3  # castigo por chocar

    if isinstance(self.grid.get(*self.agent_pos), Goal):
        reward = 10
        terminated = True

    return obs, reward, terminated, truncated, info
```