Simulación
Tema: Inteligencia Artificial 1.

Examen Final

Nombre: Walter Bau
Objetivo:

- Consolidar los conocimientos adquiridos en clase sobre la Inteligencia Artificial (IA) aplicada a juegos y búsquedas.

Enunciado:
1.Validar y probar la librearía de GYM con python utilizando algunos de los siguientes ejemplos
https://gym.openai.com/envs/#toy_text
https://towardsdatascience.com/reinforcement-learning-with-openai-d445c2c687d2
2. Describir paso a paso y comentado el código, realizar varias casos de pruebas y ejecutar.

In [47]:
```python
import gym
import numpy as np
import time

class QLearningAgent(object):
    # Inicializar Parametros
    def __init__(self, stateSpace_size, actionSpace_size, learning_rate=0.01, gamma=0.9, epsilon=0.1):
        self.stateSpace_size = stateSpace_size  #
        self.actionSpace_size = actionSpace_size  #   Tama;o del Espacio
        self.learning_rate = learning_rate  #
        self.gamma = gamma  #
        self.epsilon = epsilon  #
        self.QTable = np.zeros((stateSpace_size, actionSpace_size))  # Tabla Q
        self.runTimes = 10



    # Estable el numero de iteraciones
    def setRunTimes(self, run_times):
        self.runTimes = run_times

    # Configuraciones
    def setTrust(self, trust_ratio):
        self.epsilon = trust_ratio

    # Selecciona una estado de Accion
    def getAction(self, state):
        sand = np.random.uniform(0, 1)
        if sand <= self.epsilon:
            action = np.random.choice(self.actionSpace_size)
        else:
            action = self.chooseActionFromQTable(state)
        return action
```

Examen Final

```
    #Tabla Q
    def chooseActionFromQTable(self, state):
        Q_max = np.max(self.QTable[state, :])
        action_list = np.where(self.QTable[state, :] == Q_max)[0]
        action = np.random.choice(action_list)
        return action

    # Actualizar tabla
    def updateQTable(self, state, action, reward, next_state, done):
        if done:
            Q_future = 0
        else:
            Q_future = np.max(self.QTable[next_state, :])
        updateValue = self.learning_rate * (reward + self.gamma * Q_future - self.QTable[state, action])
        self.QTable[state, action] += updateValue

    # Mostrar Tabla
    def showQTable(self):
        print(self.QTable)

    # Leer los datos de la tabla
    def loadQTable(self, path):
        self.QTable = np.load(path)
        print(path + ' loaded.')

    #Guarda los datos de la tabla
    def saveQTable(self, path):
        np.save(path, self.QTable)
        print(path + ' saved.')
```

```
    # Codigo
    def run(self, env, agent, render):
        total_steps = 0
        while True:
            state = env.reset()  #  Restablece el entorno, es como reiniciar el juego y empieza otra iteracion
            episode_reward = 0
            episode_steps = 0
            if total_steps >= self.runTimes:
                break
            while True:
                action = self.chooseActionFromQTable(state)  # Selecciona un algoritmo de la libreria
                next_state, reward, done, info = env.step(action)  # Interactua con el escenario
                state = next_state  # Guarda el estado anteriro
                episode_reward += -reward
                episode_steps += 1  # Guarda los pasos que va a seguir el episodio
                if render:
                    time.sleep(0.5)
                    env.render()  # Renderiza un nuevo grafico
                if done:
                    total_steps += 1
                    print("Iteracion " + str(total_steps) + ", numero de pasos " + str(episode_steps) + ", premio " + str(
                        episode_reward))
                    break
```

```
render = True # valor para mostrar el entorno
env = gym.make('Taxi-v3')# se puede cambiar por FrozenLake-v0, FrozenLake8x8-v0 algoritmos de la pagian gym

agent = QLearningAgent(
        stateSpace_size=env.observation_space.n,
        actionSpace_size=env.action_space.n,
        learning_rate=0.1,
        gamma=0.9,
        epsilon=0.3)  # Define un agente

#agent.loadqtable ('./ q_table.npy') # Es para cargar una tabla

agent.showQTable()
agent.setRunTimes(1)#numero de episodios
agent.run(env, agent, render)
```

Examen Final

```
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 ...
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
+---------+
|R: | : :G|
| : |▒: : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : |▒: : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| : | :▒: |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : | :▒: |
| : : : : |
| | : | : |
|Y| : |B: |
```

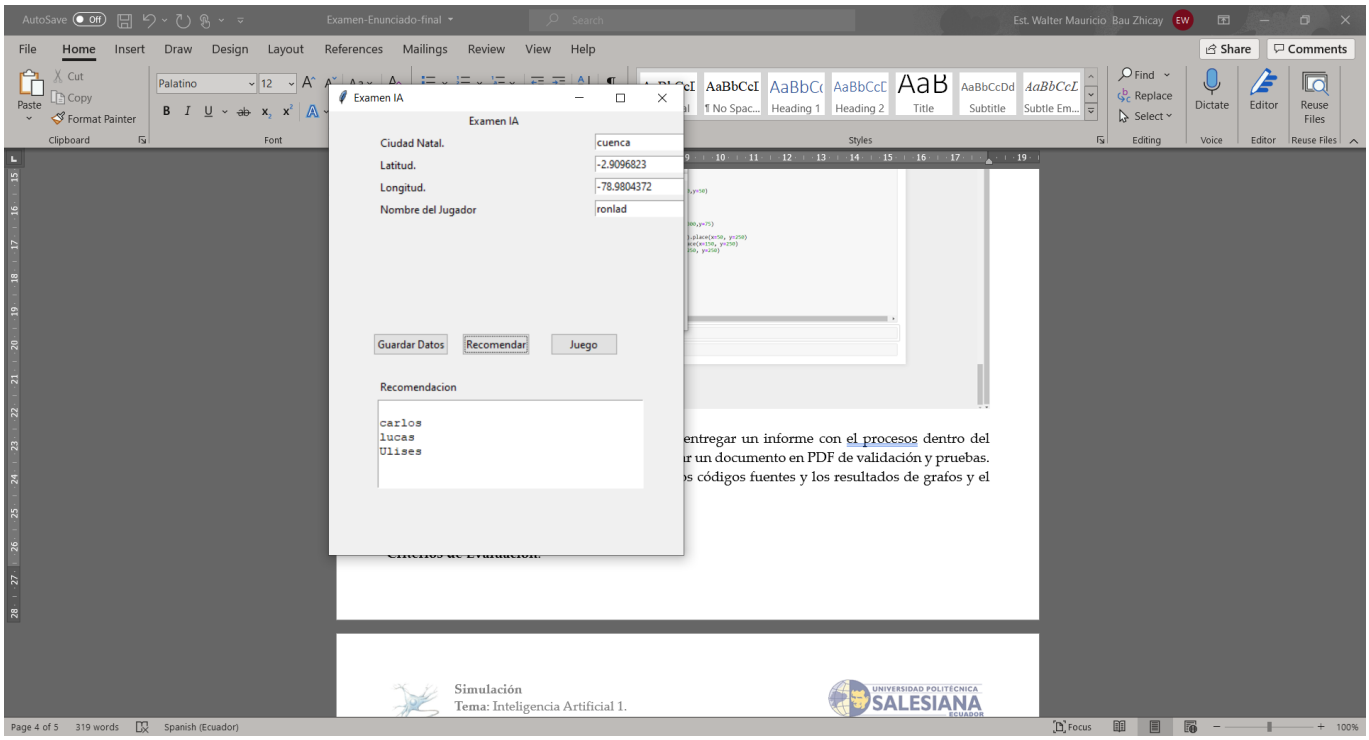
RGBY son cuatro puntos, el púrpura y el azul se cubrirán aleatoriamente con ellos, y el punto azul representa la posición del pasajero llamado automóvil, y el punto púrpura representa el destino del pasajero, el automóvil se inicializará aleatoriamente en el formulario

El entorno Toy Text en GYM contiene muchos medios, presenta tres entornos de renderizado.

3. Dentro del juego el usuario deberá escoger/ingresar su ciudad natal incluido latitud y longitud y en base a ello recomendar usuarios cercanos utilizando el algoritmos A* y Yenn, se debe tener una base de datos de al menos 50 usuarios dentro de una misma ciudad (Tomar datos de pruebas anteriores o generar una nueva base de datos), tener en presente que el árbol debe tener al menos 7 niveles o superior y con 3 conexiones de nodos cada uno.

Examen Final

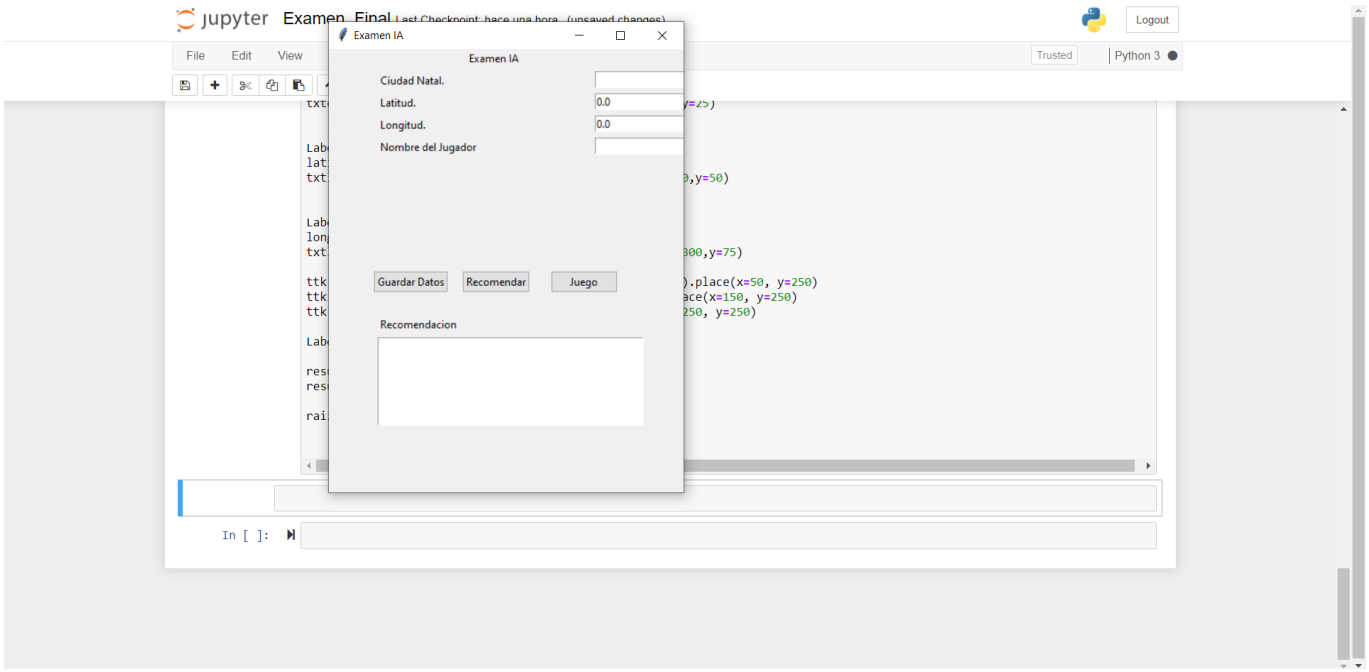
4. En base a la información proporcionada se deberá generar un sistema que permita mostrar usuarios cercanos y recomendar usuarios con los que se debe jugar o conocer.



5. Realizar el sistema con una interfaz gráfica para acceder y probar el sistema.

Simulación
Tema: Inteligencia Artificial 1.

Examen Final



Código y documentos de entrega: Se deberá entregar un informe con el procesos dentro del mismo tener capturas del uso del juego y generar un documento en PDF de validación y pruebas. Finalmente subir todo al repositorio incluido los códigos fuentes y los resultados de grafos y el juego.

Criterios de Evaluación:

- Neo4J y Búsquedas : 30%
- Juego IA: 30%
- GUI: 20%
- Informe PDF: 20%
- Usabilidad: 10%

Fecha de entrega: **02/08/2021** – 23**:55.**

Nota: Cualquier pregunta o duda con respecto al examen escribirme por correo electrónico o whatssapp.