In [ ]:

```
#Desarrollo
# Universidad Politécnica Salesiana - UPS
# Estudiante: Rafael Angamarca
# Examen Inteligencia Artificial
# Desarrollo


#Segunda Parte de Examen.
#Dentro del juego el usuario deberá escoger/ingresar su ciudad natal incluido latitud 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 conex
```

In [2]:

```
import numpy as np
import gym
import random
from neo4j import GraphDatabase

unique_transaction_types = ["MATCH (Ciudad)-[:Pertenece]-> (Pais {nombre: 'Ecuador'}) R
class Neo4jService:

    def __init__(self, uri, user, password):
        self._driver = GraphDatabase.driver(uri, auth=(user, password))

    def close(self):
        self._driver.close()

    def close(self):
        self._driver.close()

    def crear_Pais(self, tx, nombre):
        tx.run("CREATE (:Pais {nombre: $nombre})", nombre=nombre)

    def crear_Ciudad(self, tx, nombre):
        tx.run("CREATE (:Ciudad {nombre: $nombre})", nombre=nombre)

    def crear_Usuario(self, tx, nombre, latitud, longitud):
        tx.run("CREATE (:USUARIO {nombre: $nombre,latitud: $latitud,longitud: $longitud

    def crear_Score(self, tx, nombre):
        tx.run("CREATE (:Juego {nombre: $nombre})", nombre=nombre)

#realciones central - alcandes

    def crear_relacion_PaisCiudad(self, tx, nombre_P, nombre_C):
        tx.run("MATCH (a:Pais {nombre: $nombre_P}) "
               "MATCH (b:Ciudad {nombre: $nombre_C}) "
               "MERGE (b)-[:Pertenece]->(a)",
               nombre_P=nombre_P, nombre_C=nombre_C)

    def crear_relacion_CiudadUsuario(self, tx, nombre_C, nombre_U):
        tx.run("MATCH (a:Ciudad {nombre: $nombre_C}) "
               "MATCH (b:USUARIO {nombre: $nombre_U}) "
               "MERGE (b)-[:Vive]->(a)",
               nombre_C=nombre_C, nombre_U=nombre_U)
    def crear_relacion_JuegoUsuario(self, tx, score, nombre_U):
        tx.run("MATCH (a:Juego {nombre: $score}) "
```

```
                "MATCH (b:USUARIO {nombre: $nombre_U}) "
                "MERGE (a)-[:Juega]->(b)",
                score=score, nombre_U=nombre_U)

print("Ejecucion correcta")

def execute_transactions(transaction_execution_commands, return_result = False):
    data_base_connection = GraphDatabase.driver(uri = "bolt://localhost:7687", auth=("n
    session = data_base_connection.session()
    return_list = []

    for i in transaction_execution_commands:
        transaction_result = session.run(i)
        return_list = [j[0] for j in transaction_result]

    if return_result:
        return return_list
```

```
Ejecucion correcta
```

In [ ]:

```
from tkinter import *
neo4j = Neo4jService('bolt://localhost:7687', 'neo4j', 'cuenca')
def pulsar():
#aqui va neo4j
    aux = 0
    unique_transaction_results = execute_transactions(unique_transaction_types, True)
    with neo4j._driver.session() as session:
        if not unique_transaction_results:
            session.write_transaction(neo4j.crear_Usuario , nombreU.get(),latitud.get()
            session.write_transaction(neo4j.crear_Ciudad , ciudad.get())
            session.write_transaction(neo4j.crear_relacion_CiudadUsuario, ciudad.get(),
            session.write_transaction(neo4j.crear_relacion_PaisCiudad,"Ecuador",ciudad.
        else:
            for i in unique_transaction_results:
                if ciudad.get() == i:
                    aux = 1;
            if aux == 1:
                session.write_transaction(neo4j.crear_Usuario , nombreU.get(),latitud.g
                session.write_transaction(neo4j.crear_relacion_CiudadUsuario,ciudad.get
            else:
                session.write_transaction(neo4j.crear_Usuario , nombreU.get(),latitud.g
                session.write_transaction(neo4j.crear_Ciudad , ciudad.get())
                session.write_transaction(neo4j.crear_relacion_CiudadUsuario,ciudad.get
                session.write_transaction(neo4j.crear_relacion_PaisCiudad,"Ecuador",ciu

#EJEMPLO DE TAXI-V3
#En este entorno hay 4 localizaciones nombradas por diferentes letras (R, G, Y, B), el
#amarilla en la figura) tiene que coger al pasajero marcado de color Azul (R en este ca
#llevarlo a su destino (B, marcado de color violeta en este caso). Una vez el taxi suel
#en su destino, la tarea se ha completado con éxito y el episodio termina


#Cada vez que lo consigamos obtendremos una recompensa de 20 puntos por cada pasajero q
#y -1 por cada paso que damos durante el trayecto. También existe una penalización de -
#a un pasajero de forma ilegal.
def juego():
    #Empiezo el juego, donde el env busca el juego "Taxi-v3" en la libreria gym
    auxScore = 0
    env = gym.make("Taxi-v3")
    env.render()
```

```
action_size = env.action_space.n
state_size = env.observation_space.n
qtable = np.zeros((state_size, action_size))
#Definicion de variables para el juego
total_episodes = 50000        # Total Episodios
total_test_episodes = 100     # Total Test episodios
max_steps = 99                # Maximo pasos por episodio

learning_rate = 0.7           # Tasa de Aprendizaje
gamma = 0.618                 # Tasa de Descuento

# Parametros de Exploracion
epsilon = 1.0                 # Tasa de Exploracion
max_epsilon = 1.0             # Probabilidad de Exploracion al inicio
min_epsilon = 0.01            # Probabilidad minima de Exploracion
decay_rate = 0.01             # Tasa de decaimineto exponencial para el probleema d

# Bucle para repetir por el numero de episodios
for episode in range(total_episodes):
    # Reestablecer el environment
    state = env.reset()
    step = 0
    done = False

    for step in range(max_steps):
        # Elijimos una acción en los estados
        ## Declaracion de variable randomica
        exp_exp_tradeoff = random.uniform(0,1)

        ## Si el numero > es mayor a epsilon --> entra a la exploracion (tomando el
        if exp_exp_tradeoff > epsilon:
            action = np.argmax(qtable[state,:])

        # Caso contrario hace una eleccion al azar --> entra a la exploracion
        else:
            action = env.action_space.sample()

        new_state, reward, done, info = env.step(action)

        # Operacion para actualizar con ecuacion de Q-Learning Q(s,a):= Q(s,a) + lr
        qtable[state, action] = qtable[state, action] + learning_rate * (reward + g
                                    np.max(qtable[new_state, :]) - qtable[state, ac

        # Nuevo Estado
        state = new_state

        # Si hace el done, finaliza el episodio
        if done == True:
            break

    # Cada vez que hacemos un episodio reducimos el epsilon (porque cada vez necesi
    epsilon = min_epsilon + (max_epsilon - min_epsilon)*np.exp(-decay_rate*episode)

env.reset()
rewards = []

for episode in range(total_test_episodes):
    state = env.reset()
    step = 0
    done = False
    total_rewards = 0
```

```
        print("**************************************************")
        print("EPISODE ", episode)
        for step in range(max_steps):
            env.render()
            # Realice la acción que tenga la recompensa futura máxima esperada dado ese
            action = np.argmax(qtable[state,:])
            new_state, reward, done, info = env.step(action)
            total_rewards += reward
            if done:
                rewards.append(total_rewards)
                print ("Score", total_rewards)
                break
            state = new_state
    env.close()
    print ("Score Fínal: " +  str(sum(rewards)/total_test_episodes))
    auxScore = sum(rewards)/total_test_episodes
    with neo4j._driver.session() as session:
        session.write_transaction(neo4j.crear_Score , auxScore)
        session.write_transaction(neo4j.crear_relacion_JuegoUsuario, auxScore, nombreU.

ventana=Tk()
ventana.geometry('600x200')

etiqueta=Label(ventana,text='Ciudad:')
etiqueta.place(x=20,y=20)

etiqueta=Label(ventana,text='Usuario:')
etiqueta.place(x=20,y=60)

etiqueta=Label(ventana,text='Latitud:')
etiqueta.place(x=20,y=100)

etiqueta=Label(ventana,text='longitud:')
etiqueta.place(x=20,y=140)

boton=Button(ventana,text='Guardar',command=pulsar)
boton.place(x=300,y=60)
boton2=Button(ventana,text='Jugar',command=juego)
boton2.place(x=300,y=100)

nombreU=StringVar()
ciudad=StringVar()
latitud=StringVar()
longitud=StringVar()

cajatexto=Entry(ventana,textvariable=ciudad)
cajatexto.place(x=100,y=20)

cajatexto=Entry(ventana,textvariable=nombreU)
cajatexto.place(x=100,y=60)

cajatexto1=Entry(ventana,textvariable=latitud)
cajatexto1.place(x=100,y=100)

cajatexto2=Entry(ventana,textvariable=longitud)
cajatexto2.place(x=100,y=140)


ventana.mainloop()
```

```
+---------+
```

```
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

**************************************************
EPISODE  0
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)
+---------+
|R: | : :G|
|_: | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| :_: : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : | : : |
| : :_: : |
```