**UNIVERSIDAD POLITECNICA SALESIANA**

**EXAMEN DE INTELIGENCIA ARTIFICIAL**

**1. Desarrollar un juego (tema libre) empleando una de las 2 siguientes alternativas:**

1. easyAI
2. Universe + GYM El juego deberá implementar algún algoritmo de IA y de igual forma, generar un informe de movimientos, puntajes y quién gana la partida. Se debe tener un juego en donde se tenga un jugar humano y otro utilizando Inteligencia Artificial, finalmente no se puede repetir el juego por más de tres personas por lo que se debe publicar en el foro el juego seleccionado.
3. Dentro del juego el usuario puede registrar he ingresar los gustos de alguna área basadas en el lugar geográfico por ejemplo: comida, películas, lugares turísticos etc.
4. En base a la información proporcionada se deberá generar un sistema que permita mostrar lugares de interés, para ello tomar los datos de las tareas y pruebas dentro de una base de datos orientadas a grafos.
5. Realizar el sistema con una interfaz gráfica y almacenar los puntajes y datos de los usuarios o jugadores.

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

Criterios de Evaluación:

• Neo4J y Búsquedas : 30%

• Juego IA: 30%

• GUI: 20%

• Informe PDF: 20%

• Usabilidad: 10%

**Inteligencia Artificial que juega automáticamente a un juego de coches.**

OpenAI

OpenAI Gym es un conjunto de herramientas para desarrollar y comparar algoritmos de aprendizaje por refuerzo. Esta es la biblioteca de código abierto del gimnasio, que le brinda acceso a un conjunto estandarizado de entornos, para su instalacion solo se ejecuta el siguiente comando:

pip install gym

**from** \_\_future\_\_ **import** division

**import** pygame

**import** random

**import** time

**from** tf\_agents.agents.dqn **import** dqn\_agent

**from** tf\_agents.drivers **import** dynamic\_step\_driver

**from** random **import** randint

**import** numpy **as** np

**from** keras.utils **import** to\_categorical

**import** sys

​

pygame.init()

​

width **=** 800 **+** 800

height **=** 600

size **=** (width, height)

fps **=** 120

counter\_games **=** 0

record **=** 0

count\_episodes **=** 0

​

screen **=** pygame.display.set\_mode(size)

clock **=** pygame.time.Clock()

​

font **=** pygame.font.Font('/Users/zhimi/Downloads/ia\_car-master/fonts/cargo.ttf', 40)

score **=** pygame.font.Font('/Users/zhimi/Downloads/ia\_car-master/fonts/cargo.ttf', 30)

font2 **=** pygame.font.Font('/Users/zhimi/Downloads/ia\_car-master/fonts/cargo.ttf', 30)

​

background **=** pygame.image.load("/Users/zhimi/Downloads/ia\_car-master/images/roadway.jpg")

backrect **=** background.get\_rect()

​

carimg **=** pygame.image.load("/Users/zhimi/Downloads/ia\_car-master/images/car.png")

car\_width **=** 49

​

truckimg **=** pygame.transform.scale(pygame.image.load("/Users/zhimi/Downloads/ia\_car-master/images/pickup.png"), (70, 145))

​

​

**def** avoided(count):

scoreFont **=** score.render("Score: %d" **%** count, **True**, (0, 0, 0))

screen.blit(scoreFont, (50, 570))

​

​

**def** print\_record(record):

scoreFont **=** score.render("Record: %d" **%** record, **True**, (0, 255, 0))

screen.blit(scoreFont, (450, 570))

​

​

**def** print\_epochs(count\_episodes):

scoreFont **=** score.render("Epochs: %d" **%** count\_episodes, **True**, (255, 0, 0))

screen.blit(scoreFont, (50, 10))

​

​

**def** print\_training(train):

**if** train:

scoreFont **=** score.render("Training", **True**, (255, 255, 255))

**else**:

scoreFont **=** score.render("Playing", **True**, (255, 255, 255))

screen.blit(scoreFont, (450, 10))

​

​

**def** truck(truck\_x, truck\_y):

screen.blit(truckimg, (truck\_x, truck\_y))

​

​

**def** car(x, y):

**if** x **<=** width **//** 2:

screen.blit(carimg, (x, y))

​

​

**def** message2(x):

messageFont2 **=** font.render("CHOCASTES", **True**, (0, 0, 0))

rect **=** messageFont2.get\_rect()

rect.center **=** ((width **//** 4), (height **//** 2))

screen.blit(messageFont2, rect)

pygame.display.update()

time.sleep(0.2)

playing(params, agent)

​

​

**def** message(x):

messageFont **=** font.render("FUERA DEL CAMINO", **True**, (0, 0, 0))

rect **=** messageFont.get\_rect()

rect.center **=** ((width **//** 4), (height **//** 2))

screen.blit(messageFont, rect)

pygame.display.update()

time.sleep(0.2)

playing(params, agent)

​

​

**def** crashed2():

message2("CHOCASTE")

​

​

**def** crashed():

message("FUERA DEL CAMINO")

​

​

**def** get\_random\_pos():

truc\_pos **=** random.randint(0, 3)

**if** truc\_pos **==** 0:

truck\_x **=** 90

**elif** truc\_pos **==** 1:

truck\_x **=** 280

**elif** truc\_pos **==** 2:

truck\_x **=** 470

**elif** truc\_pos **==** 3:

truck\_x **=** 660

**return** truck\_x

​

​

*# RED NEURONAL #*

sep\_ver **=** 60

sep\_hor **=** 100

​

screen.fill((255, 255, 255))

iz\_font **=** font2.render("Izquierda", **True**, (0, 0, 0))

ent\_font **=** font2.render("Entrada", **True**, (0, 0, 0))

na\_font **=** font2.render("Nada", **True**, (0, 0, 0))

der\_font **=** font2.render("Derecha", **True**, (0, 0, 0))

screen.blit(ent\_font, (width **//** 2 **+** 40, 30))

screen.blit(iz\_font, (width **//** 2 **+** 600, 175 **+** (int(0 **\*** sep\_ver **\*** 1.5))))

screen.blit(na\_font, (width **//** 2 **+** 600, 175 **+** (int(1 **\*** sep\_ver **\*** 1.5))))

screen.blit(der\_font, (width **//** 2 **+** 600, 175 **+** (int(2 **\*** sep\_ver **\*** 1.5))))

​

co1 **=** font2.render("C.O. 1", **True**, (0, 0, 0))

co2 **=** font2.render("C.O. 2", **True**, (0, 0, 0))

sal **=** font2.render("Salida", **True**, (0, 0, 0))

screen.blit(co1, (width **//** 2 **+** 260, 20))

screen.blit(co2, (width **//** 2 **+** 410, 90))

screen.blit(sal, (width **//** 2 **+** 530, 120))

​

**def** draw\_nn(we, arg\_max, state\_old):

we **=** np.array([(w **+** 1) **/** 2 **for** w **in** we])

we\_0, we\_1, we\_2, we\_3, we\_4 **=** we[0], we[1], we[2], we[3], we[4]

we\_0 **=** (we\_0 **-** np.min(we\_0)) **/** np.ptp(we\_0)

we\_1 **=** (we\_1 **-** np.min(we\_1)) **/** np.ptp(we\_1)

we\_2 **=** (we\_2 **-** np.min(we\_2)) **/** np.ptp(we\_2)

we\_3 **=** (we\_3 **-** np.min(we\_3)) **/** np.ptp(we\_3)

we\_4 **=** (we\_4 **-** np.min(we\_4)) **/** np.ptp(we\_4)

​

**try**:

**for** i **in** range(len(state\_old)):

pygame.draw.circle(screen, (state\_old[i] **\*** 255, 100, 0), (width **//** 2 **+** 80, 80 **+** (int(i **\*** sep\_ver**/**2))), 10)

**for** j **in** range(len(we\_0[i])):

pygame.draw.line(screen,

(we\_0[i][j] **\*** 255, 100, 0), (width **//** 2 **+** 80 **+** 20, 80 **+** (int(i **\*** sep\_ver**/**2))),

(width **//** 2 **+** 300 **-** 20, 80 **+** (int(j **\*** sep\_ver))), 2)

**for** i **in** range(len(we\_1)):

pygame.draw.circle(screen, (we\_1[i] **\*** 255, 0, 0), (width **//** 2 **+** 300, 80 **+** (i **\*** sep\_ver)), 20)

**for** j **in** range(len(we\_2[i])):

pygame.draw.line(screen,

(we\_2[i][j] **\*** 255, 0, 0), (width **//** 2 **+** 300 **+** 20, 80 **+** (i **\*** sep\_ver)),

(width **//** 2 **+** 450 **-** 20, 150 **+** (int(j **\*** sep\_ver **\*** 1.5))), 2)

**for** i **in** range(len(we\_3)):

pygame.draw.circle(screen, (0, 0, we\_3[i] **\*** 255), (width **//** 2 **+** 450, 150 **+** (int(i **\*** sep\_ver **\*** 1.5))), 20)

**for** j **in** range(len(we\_4[i])):

pygame.draw.line(screen,

(0, 0, we\_4[i][j] **\*** 255), (width **//** 2 **+** 450 **+** 20, 150 **+** int(i **\*** sep\_ver **\*** 1.5)),

(width **//** 2 **+** 570 **-** 20, 190 **+** (int(j **\*** sep\_ver **\*** 1.5))), 2)

**for** i **in** range(3):

**if** i **==** arg\_max:

pygame.draw.circle(screen, (0, 255, 0), (width **//** 2 **+** 570, 190 **+** (int(i **\*** sep\_ver **\*** 1.5))), 20)

**else**:

pygame.draw.circle(screen, (150, 150, 150), (width **//** 2 **+** 570, 190 **+** (int(i **\*** sep\_ver **\*** 1.5))), 20)

**except**:

**pass**

​

​

**def** playing(params, agent):

x **=** get\_random\_pos()

y **=** 480

*# screen.fill((255, 255, 255))*

​

**global** counter\_games

**global** record

**global** count\_episodes

​

weights\_filepath **=** params['weights\_path']

**if** params['load\_weights']:

agent.model.load\_weights(weights\_filepath)

​

truck\_x **=** get\_random\_pos()

​

truck\_y **=** **-**145

truck\_speed **=** 10

truck\_height **=** 145

truck\_width **=** 102 **/** 2

score **=** 0

arg\_max **=** 1

​

**while** params['episodes'] **>=** count\_episodes:

pygame.event.get()

*# clock.tick(fps)*

​

**if** **not** params['train']:

agent.epsilon **=** 0

**else**:

*# agent.epsilon is set to give randomness to actions*

agent.epsilon **=** 1 **-** (count\_episodes **\*** params['epsilon\_decay\_linear'])

**if** agent.epsilon **<** 0.01:

agent.epsilon **=** 0.01

​

*# get old state*

state\_old **=** agent.get\_state(height, x, y, truck\_x, truck\_y, arg\_max, car\_width, truck\_width)

*# rand\_var = random.random()*

*# print(agent.epsilon, rand\_var, counter\_games)*

*# if randint(0, 1) < agent.epsilon:*

**if** random.random() **<** agent.epsilon:

arg\_max **=** randint(0, 2)

final\_move **=** to\_categorical(arg\_max, num\_classes**=**3)

**else**:

*# predict action based on the old state*

prediction **=** agent.model.predict(state\_old.reshape((1, **-**1)))

arg\_max **=** np.argmax(prediction[0])

final\_move **=** to\_categorical(arg\_max, num\_classes**=**3)

​

**if** arg\_max **==** 0:

xChange **=** **-**190

**elif** arg\_max **==** 1:

xChange **=** 0

**else**:

xChange **=** **+**190

​

x **+=** xChange

​

screen.blit(background, backrect)

​

truck(truck\_x, truck\_y)

truck\_y **+=** truck\_speed

​

car(x, y)

​

state\_new **=** agent.get\_state(height, x, y, truck\_x, truck\_y, arg\_max, car\_width, truck\_width)

​

avoided(score)

**if** score **>=** record:

record **=** score

​

print\_record(record)

print\_training(params['train'])

print\_epochs(count\_episodes)

​

crash\_flag **=** **False**

reward **=** 0

​

*# Pintamos la red con los diferentes pesos del modelo y la salida obtenida*

draw\_nn(agent.model.weights, arg\_max, state\_old)

​

*# crash detection if the car goes off the road*

**if** x **>** (width **//** 2 **-** 87) **or** x **<** 35:

reward **=** **-**10

**if** params['train']:

agent.replay\_new(agent.memory, params['batch\_size'])

crash\_flag **=** **True**

counter\_games **=** 0

count\_episodes **+=** 1

​

*# starting the truck along random coordinates*

**if** truck\_y **>** height:

**if** counter\_games **==** 0:

reward **=** 0

**else**:

reward **=** 1 *# Great prize :)*

truck\_y **=** **-**145

truck\_x **=** get\_random\_pos()

*# if params['train']:*

score **+=** 1 *# increase the score +1 for every truck is avoided*

truck\_speed **+=** 1 *# .2 # increase the speed by 0.2 for every truck passed*

count\_episodes **+=** 1

​

*# collision detection for hitting the truck*

**if** y **<** truck\_y **+** 145:

**if** x **>** truck\_x **and** x **<** truck\_x **+** truck\_width **or** x **+** car\_width **>** truck\_x **and** x **+** car\_width **<** truck\_x **+** truck\_width:

reward **=** **-**10

**if** params['train']:

agent.replay\_new(agent.memory, params['batch\_size'])

crash\_flag **=** **True**

counter\_games **=** 0

count\_episodes **+=** 1

​

**if** params['train']:

*# train short memory base on the new action and state*

agent.train\_short\_memory(state\_old, final\_move, reward, state\_new, crash\_flag)

*# store the new data into a long term memory*

agent.remember(state\_old, final\_move, reward, state\_new, crash\_flag)

​

pygame.display.flip()

counter\_games **+=** 1

​

**if** crash\_flag:

crashed2()

​

**if** params['episodes'] **==** count\_episodes:

agent.model.save\_weights(params['weights\_path'])

sys.exit()

​

​

**def** define\_parameters():

params **=** dict()

params['epsilon\_decay\_linear'] **=** 1 **/** 100

params['learning\_rate'] **=** 0.001

params['first\_layer\_size'] **=** 10 *# neurons in the first layer*

params['second\_layer\_size'] **=** 5 *# neurons in the second layer*

params['episodes'] **=** 500

params['memory\_size'] **=** 10000

params['batch\_size'] **=** 1000

params['weights\_path'] **=** 'weights\_car.hdf5'

params['train'] **=** **False**

**if** params['train']:

params['load\_weights'] **=** **False**

**else**:

params['load\_weights'] **=** **True**

**return** params

​

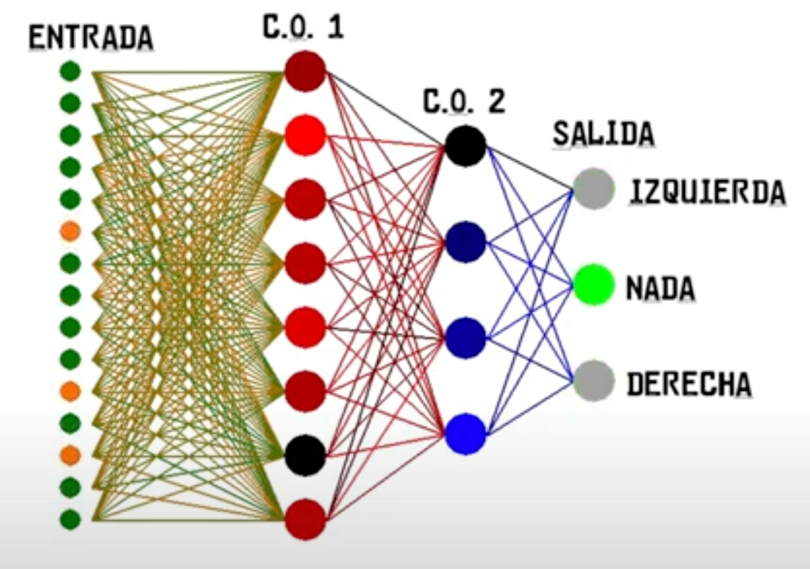
​

**if** \_\_name\_\_ **==** '\_\_main\_\_':

params **=** define\_parameters()

agent **=** QLAgent(params)

playing(params, agent)

​

**SISTEMA DE RECOMENDACION DE PARQUES**

**from** neo4j **import** GraphDatabase

​

**class** Neo4jService(object):

​

**def** \_\_init\_\_(self, uri, user, password):

self.\_driver **=** GraphDatabase.driver(uri, auth**=**(user, password))

​

**def** close(self):

self.\_driver.close()

**def** crear\_nodo(self, tx, nombre,pre):

tx.run("MERGE (jugador:Person {name:$nombre})"

"SET jugador.embedding = [$pre]",nombre**=**nombre

,pre**=**pre)

**def** recomendacion(self,tx):

result **=** tx.run("MATCH (m:Movie)\n"

"WITH {item:id(m), weights: m.embedding} AS userData\n"

"WITH collect(userData) AS data\n"

"CALL gds.alpha.similarity.pearson.stream({\n"

"data: data,\n"

"skipValue: null\n"

"})\n"

"YIELD item1, item2, similarity\n"

"RETURN gds.util.asNode(item1).name AS from, gds.util.asNode(item2).name AS to, similarity\n"

"ORDER BY similarity DESC")

**for** record **in** result:

r1**=**(record["from"])

r2**=**(record["to"])

r3**=**(record["similarity"])

**if** r1 **==**nombre.get() **and** r3**>=**0.80:

resultado.insert(tk.END, "\n"**+**r2)

In [7]:

**from** tkinter **import** **\***

**from** tkinter **import** ttk

**from** tkinter **import** messagebox

**import** tkinter **as** tk

​

vent **=** Tk()

​

**def** crear():

neo4j **=** Neo4jService('bolt://localhost:7687', 'neo4j', 'final')

**with** neo4j.\_driver.session() **as** session:

session.write\_transaction(neo4j.crear\_nodo , nombre.get(),float(pr.get()))

**def** listar():

neo4j **=** Neo4jService('bolt://localhost:7687', 'neo4j', 'final')

**with** neo4j.\_driver.session() **as** session:

session.read\_transaction(neo4j.recomendacion)

vent.geometry('400x400')

vent.title('Examen')

​

Label(vent, text**=**"Nuevo Usuario:").place(x**=**55, y**=**15)

nombre **=** ttk.Entry(vent)

nombre.place(x **=** 150, y**=**15)

ttk.Button(vent, text**=**'Crear Nuevo Usuario', command**=**crear).place(x**=**150, y**=**55)

​

Label(vent, text**=**"Prefieres jugar en un parque con arboles?").place(x**=**55, y**=**100)

pr **=** Spinbox(vent, from\_**=**0, to**=**1, width**=**5, increment**=**1)

pr.place(x**=**320, y**=**100)

ttk.Button(vent, text**=**'Recomendar', command**=**listar).place(x**=**200, y**=**130)

​

Label(vent, text**=**"Se recomienda visitar estos lugares.").place(x**=**80, y**=**160)

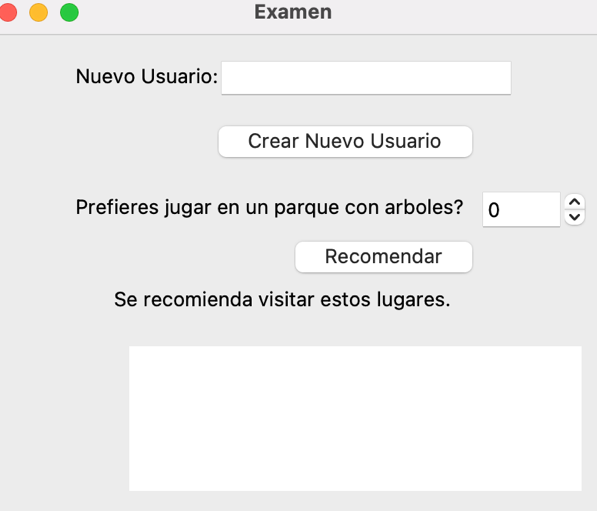
resultado **=** Text(vent)

resultado.place(x **=** 90, y**=**200, width**=**300, height**=**100)

​

vent.mainloop()

Resultados:

Los nuevos Usuarios son creados con exito

