FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES

CARRERA: COMPUTACIÓN/INGENIERÍA DE SISTEMAS	**ASIGNATURA**: SISTEMAS EXPERTOS - INTELIGENCIA ARTIFICIAL 1

NRO. PROYECTO:	1.1	**TÍTULO PROYECTO**: Proyecto Integrador Interciclo Desarrollo e implementación de un sistema de busqueda, casos y/o similitud en una base de datos orientada a grafos.

OBJETIVO:

Reforzar los conocimientos adquiridos en clase sobre la búsqueda, casos y/o similitud para el diseño e implementación de sistemas inteligentes utilizando redes sociales en el contexto político del Ecuador.

INSTRUCCIONES:	**1.** Revisar el contenido teórico del tema
	2. Profundizar los conocimientos revisando los libros guías, los enlaces contenidos en los objetos de aprendizaje, cuadernos Python y la documentación disponible en fuentes académicas en línea.
	3. Deberá desarrollar un sistema inteligente en base a redes sociales, por ejemplo twitter del presidente con mas seguidores, entre otros.
	4. Deberá generar un informe empleando una herramienta Web 2.0 (Tutorial o manual técnico) .
	5. Tomar en consideración que la evaluación del trabajo a realizarse de forma **individual** dependerá de los siguientes parámetros: Nivel de precisión y explicación de la propuesta planteada en cada uno de los algoritmos de busqueda, similitud y casos (Cuadernos Jupyter). **50%** Tutorial del sistema del uso de los algoritmos en Neo4j **25%** (Pagina Web) Exposición **25%.** **6. Fecha de presentación: 08 de Junio** del 2021 a 23:55. **7. Puntos extras:** Cualquier mejora, innovación o investigación adicional sera valorado como puntos extras directos al inter-ciclo.

ACTIVIDADES POR DESARROLLAR

1. Investigue**,** diseñe y desarrolle e implemente 1 y/o 2 algoritmos dentro de una base de datos orientadas a grafos del siguiente link (https://neo4j.com/docs/graph-data-science/current/algorithms/).

Problema: Se desea generar métodos inteligentes para encontrar información basada en un ejemplo aplicando datos del presidente obtenidos de redes sociales, etc..

Pasos a seguir:
1. Realizar una extracción de datos en base a palabras claves del presidente.
2. Migrar esta información a la base de grafos Neo4j, en virtud de ello, se deberá tener al menos 2000 nodos.
3. Implementar los algoritmos inteligentes dentro de la base de grafos.
4. Consultar información y/o resultados del grafo aplicando los algoritmos inteligentes.
6. Agregar conclusiones y recomendaciones.

Introducción:
Los algoritmos de grafos se utilizan para calcular métricas para grafos, nodos o relaciones.
Pueden proporcionar información sobre entidades relevantes en el gráfico (centralidades, clasificación) o estructuras inherentes como las comunidades (detección de comunidad, partición de grafos, agrupación, etc.).
Muchos algoritmos de gráficos son enfoques iterativos que frecuentemente atraviesan el grafo para el cálculo utilizando caminatas aleatorias, búsquedas de amplitud o de profundidad, o coincidencia de patrones. En Neo4j Graph Data Science contiene una gran cantidad de algoritmos que se pueden aplicar sobre una base de datos orientadas a grafos:

- Sección 5.2, "Algoritmos de centralidad"
- Sección 5.3, "Algoritmos de detección de la comunidad"
- Sección 5.4, "Algoritmos de similitud"
- Sección 5.6, "Algoritmos de predicción de enlaces"

La **solución** se debe seleccionar 1 algoritmo (IA) y 2(SE) y generar un cuaderno que contenga la siguiente información:

- Introducción al tipo de algoritmo.
- Descripción del algoritmo.
- Ejemplificación usando datos reales (Redes Sociales).
- Resultados y análisis.
- Mejoras y recomendaciones(Material adicional o refuerzo del algoritmo).
- Conclusiones y trabajos futuros(Ejercicios/Áreas aplicables).

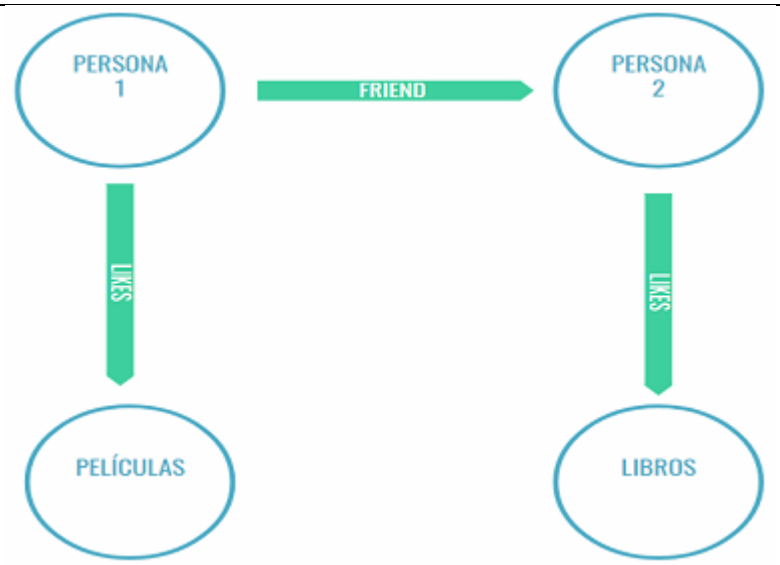
2. Tutorial técnico del uso y proceso de Neo4j (Manual técnico):
- Generar una pagina web:
    - Planteamiento y descripción del problema.
        - Proceso de solución.
    - Conclusiones y recomendaciones.
    - Resultados y validaciones.

Resolución

Base de datos grafica diseñada para tratar las relaciones entre los datos. Su objetivo es mantener los datos restringidos a un modelo predefinido.

Los datos se almacenan en la forma que se los obtiene. mostrando así a cada entidad individual que se conecta o se relacion con otras entidades

NODOS Y RELACIONES



Dentro de Neo4j existe una gran variedad de algoritmos inteligentes cuyo propósito es agilizar el procesamiento de grandes cantidades de datos y mediante grafos visualizar de mejor manera los mismos.

La biblioteca de Neo4j Graph Data Science (GDS) contiene muchos algoritmos de gráficos. Los algoritmos se dividen en categorías que representan diferentes clases de problemas.
Los algoritmos existen en uno de los tres niveles de madurez:

Calidad de la producción

- Indica que el algoritmo ha sido probado en cuanto a estabilidad y escalabilidad.
- Los algoritmos de este nivel tienen el prefijo gds.<algorithm>.

· Beta

- Indica que el algoritmo es candidato para el nivel de calidad de producción.
- Los algoritmos de este nivel tienen el prefijo gds.beta.<algorithm>.

· Alfa

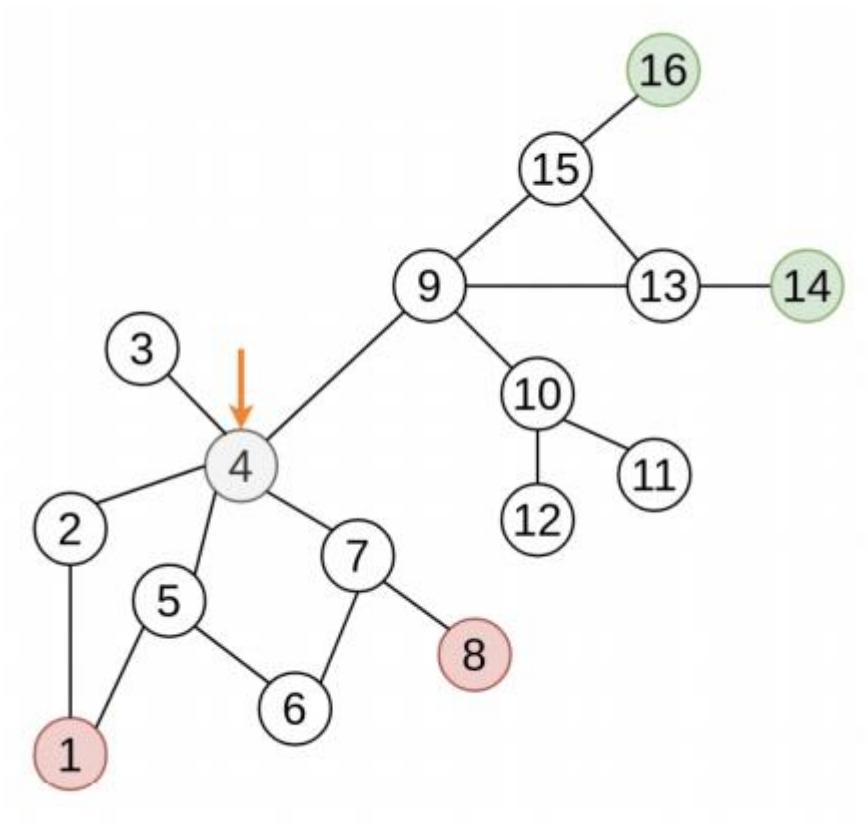
- o Indica que el algoritmo es experimental y se puede cambiar o eliminar en cualquier momento.
- o Los algoritmos de este nivel tienen el prefijo gds.alpha.<algorithm>.

Se dividen en los siguientes:

- · Algoritmos de centralidad.
- · Algoritmos de detección de comunidades.
- · Algoritmos de similitud.
- · Algoritmos de búsqueda de rutas.
- · Algoritmos de predicción de enlaces.
- · Nodos embebidos.

Para este trabajo se hizo uso del algoritmo de detección de comunidades.

COMMUNITY DETECTION ALGORITHMS



LABEL PROPAGATION

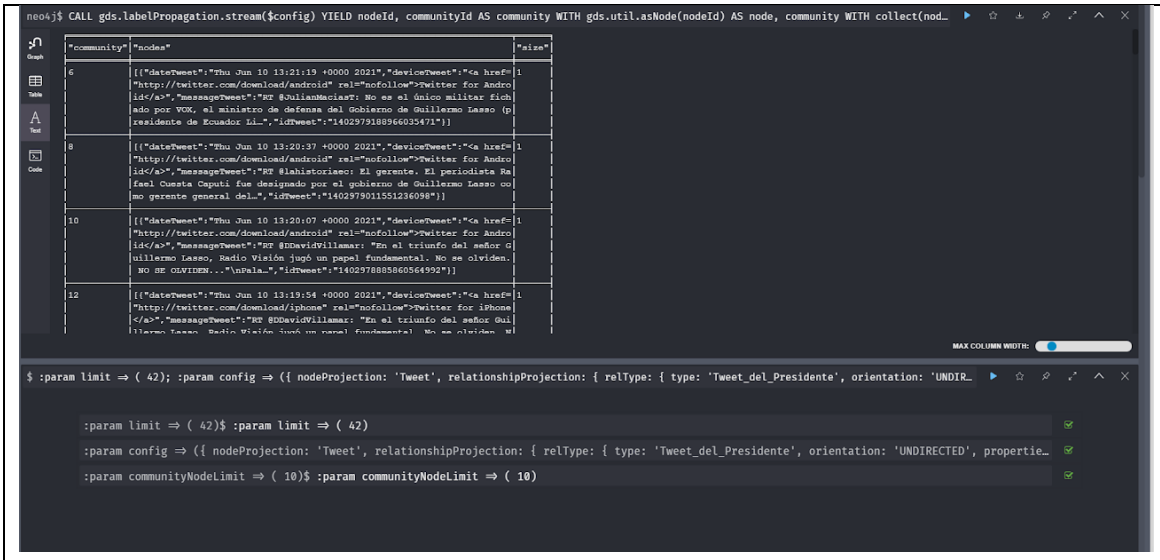
Las redes sociales se han extendido por todo el mundo y están creciendo día a día. Considere una red de medios sociales en la que conozca los intereses de algunas personas y desee predecir los intereses de otras para que podamos orientar las campanas de marketing. Para este propósito, podemos utilizar la técnica de aprendizaje automático semi-supervisado basada en gráficos llamada Label Propagation.

MPLEMENTAR LOS ALGORITMOS INTELIGENTES DENTRO DE LA BASE DE GRAFOS.

APLICACIÓN DEL ALGORITMO LABEL PROPAGATION

Utilizando la Base de datos tratamos de buscar un tipo de comunidad
A continuación se muestra el código del algoritmo a utilizar

RESULTADOS OBTENIDOS


neo4j$ CALL gds.labelPropagation.stream($config) YIELD nodeId, communityId AS community WITH gds.util.asNode(nodeId) AS node, community WITH collect(nod…
"community"
"nodes"
"size"
MAX COLUMN WIDTH:
$:param limit ⇒ (42); :param config ⇒ ({ nodeProjection: 'Tweet', relationshipProjection: { relType: { type: 'Tweet_del_Presidente', orientation: 'UNDIR…
:param limit ⇒ (42)$:param limit ⇒ (42)
:param config ⇒ ({ nodeProjection: 'Tweet', relationshipProjection: { relType: { type: 'Tweet_del_Presidente', orientation: 'UNDIRECTED', propertie…
:param communityNodeLimit ⇒ (10)$:param communityNodeLimit ⇒ (10)


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
neo4j$ CALL gds.labelPropagation.stream($config) YIELD nodeId, communityId AS community WITH gds.util.asNode(nodeId) AS node, community WITH collect(node) AS allN…
*(42)
Tweet(42)
Displaying 42 nodes, 0 relationships.

In [ ]:
```
#simularemos el ambiente del juego y su compotamiento en la Jupyter Notebook.
# agente será el “player 1” y sus acciones posible son 2:

#mover hacia arriba
#mover hacia abajo
#Y las reglas del juego:

#El agente tiene 3 vidas.
#Si pierde… castigo, restamos 10 puntos.
#Cada vez que le demos a la bola, recompensa, sumamos 10.
#Para que no quede jugando por siempre, limitaremos el juego a
#3000 iteraciones máximo ó
#alcanzar 1000 puntos y habremos ganado.
```

In [ ]:
```


#Proyecto Interciclo

import numpy as np
import matplotlib.pyplot as plt
from random import randint
from time import sleep
from IPython.display import clear_output
from math import ceil,floor

%matplotlib inline
```

In [ ]:
```
#Class Agente
#La clase Agente
#Dentro de la clase Agente encontraremos la tabla donde iremos almacenando las pa

#La posición actual del jugador.
#La posición “y” de la pelota.
#La posición en el eje “x” de la pelota.
#Además en esta clase, definiremos el factor de descuento, el learning rate y el

#Los métodos más importantes:

#get_next_step() decide la siguiente acción a tomar en base al ratio de exploraci
#update() aquí se actualizan las políticas mediante la ecuación de Bellman que vi
```

In [2]:
```
class PongAgent:

    def __init__(self, game, policy=None, discount_factor = 0.1, learning_rate =

        # Creamos la tabla de politicas
        if policy is not None:
            self._q_table = policy
        else:
            position = list(game.positions_space.shape)
            position.append(len(game.action_space))
            self._q_table = np.zeros(position)

        self.discount_factor = discount_factor
        self.learning_rate = learning_rate
        self.ratio_explotacion = ratio_explotacion

    def get_next_step(self, state, game):

        # Damos un paso aleatorio...
        next_step = np.random.choice(list(game.action_space))

        # o tomaremos el mejor paso...
        if np.random.uniform() <= self.ratio_explotacion:
            # tomar el maximo
            idx_action = np.random.choice(np.flatnonzero(
                    self._q_table[state[0],state[1],state[2]] == self._q_table[st
                ))
            next_step = list(game.action_space)[idx_action]

        return next_step

    # actualizamos las politicas con las recompensas obtenidas
    def update(self, game, old_state, action_taken, reward_action_taken, new_stat
        idx_action_taken =list(game.action_space).index(action_taken)

        actual_q_value_options = self._q_table[old_state[0], old_state[1], old_st
        actual_q_value = actual_q_value_options[idx_action_taken]

        future_q_value_options = self._q_table[new_state[0], new_state[1], new_st
        future_max_q_value = reward_action_taken  +  self.discount_factor*future_
        if reached_end:
            future_max_q_value = reward_action_taken #maximum reward

        self._q_table[old_state[0], old_state[1], old_state[2], idx_action_taken]
                                                self.learning_rate*(future_max_q_va

    def print_policy(self):
        for row in np.round(self._q_table,1):
            for column in row:
                print('[', end='')
                for value in column:
                    print(str(value).zfill(5), end=' ')
                print('] ', end='')
            print('')

    def get_policy(self):
```

```
        return self._q_table
```

In [ ]:

In [ ]:
```
#Class ENVIROMENT

#En la clase de Ambiente encontramos implementada la lógica y control del juego d
#Se controla que la pelotita rebote, que no se salga de la pantalla y se encuentr

#Por Defecto se define una pantalla de 40 pixeles x 50px de alto y si utilizamos
#nos quedará definida nuestra tabla de políticas en 8 de alto y 10 de ancho (por
#Estos valores se pueden modificar!

#Además, muy importante, tenemos el control de cuándo dar las recompensas y penal
#al perder cada vida y detectar si el juego a terminado
```

In [3]:
```
class PongEnvironment:

    def __init__(self, max_life=3, height_px = 40, width_px = 50, movimiento_px =

        self.action_space = ['Arriba','Abajo']

        self._step_penalization = 0

        self.state = [0,0,0]

        self.total_reward = 0

        self.dx = movimiento_px
        self.dy = movimiento_px

        filas = ceil(height_px/movimiento_px)
        columnas = ceil(width_px/movimiento_px)

        self.positions_space = np.array([[[0 for z in range(columnas)]
                                                for y in range(filas)]
                                                   for x in range(filas)])


        self.lives = max_life
        self.max_life=max_life

        self.x = randint(int(width_px/2), width_px)
        self.y = randint(0, height_px-10)

        self.player_alto = int(height_px/4)

        self.player1 = self.player_alto  # posic. inicial del player

        self.score = 0

        self.width_px = width_px
        self.height_px = height_px
        self.radio = 2.5

    def reset(self):
        self.total_reward = 0
        self.state = [0,0,0]
        self.lives = self.max_life
        self.score = 0
        self.x = randint(int(self.width_px/2), self.width_px)
        self.y = randint(0, self.height_px-10)
        return self.state

    def step(self, action, animate=False):
        self._apply_action(action, animate)
        done = self.lives <=0 # final
        reward = self.score
        reward += self._step_penalization
        self.total_reward += reward
        return self.state, reward , done

    def _apply_action(self, action, animate=False):
```

```
        if action == "Arriba":
            self.player1 += abs(self.dy)
        elif action == "Abajo":
            self.player1 -= abs(self.dy)

        self.avanza_player()

        self.avanza_frame()

        if animate:
            clear_output(wait=True);
            fig = self.dibujar_frame()
            plt.show()

        self.state = (floor(self.player1/abs(self.dy))-2, floor(self.y/abs(self.d

    def detectaColision(self, ball_y, player_y):
        if (player_y+self.player_alto >= (ball_y-self.radio)) and (player_y <= (b
            return True
        else:
            return False

    def avanza_player(self):
        if self.player1 + self.player_alto >= self.height_px:
            self.player1 = self.height_px - self.player_alto
        elif self.player1 <= -abs(self.dy):
            self.player1 = -abs(self.dy)

    def avanza_frame(self):
        self.x += self.dx
        self.y += self.dy
        if self.x <= 3 or self.x > self.width_px:
            self.dx = -self.dx
            if self.x <= 3:
                ret = self.detectaColision(self.y, self.player1)

                if ret:
                    self.score = 10
                else:
                    self.score = -10
                    self.lives -= 1
                    if self.lives>0:
                        self.x = randint(int(self.width_px/2), self.width_px)
                        self.y = randint(0, self.height_px-10)
                        self.dx = abs(self.dx)
                        self.dy = abs(self.dy)
        else:
            self.score = 0

        if self.y < 0 or self.y > self.height_px:
            self.dy = -self.dy

    def dibujar_frame(self):
        fig = plt.figure(figsize=(5, 4))
        a1 = plt.gca()
        circle = plt.Circle((self.x, self.y), self.radio, fc='slategray', ec="bla
```

```
        a1.set_ylim(-5, self.height_px+5)
        a1.set_xlim(-5, self.width_px+5)

        rectangle = plt.Rectangle((-5, self.player1), 5, self.player_alto, fc='gc
        a1.add_patch(circle);
        a1.add_patch(rectangle)
        #a1.set_yticklabels([]);a1.set_xticklabels([]);
        plt.text(4, self.height_px, "SCORE:"+str(self.total_reward)+"  LIFE:"+str
        if self.lives <=0:
            plt.text(10, self.height_px-14, "GAME OVER", fontsize=16)
        elif self.total_reward >= 1000:
            plt.text(10, self.height_px-14, "YOU WIN!", fontsize=16)
        return fig
```

In [ ]:

In [ ]:
```
#fINalmente definimos una función para jugar, donde indicamos la cantidad de vece
la simulación del juego e iremos almacenando algunas estadísticas sobre el compor
si mejora el puntaje con las iteraciones y el máximo puntaje alcanzado.
```

In [ ]:
```
#jUEGO
"""""""# Para entrenar ejecutamos la función con los siguientes parámetros:

#6000 partidas jugará
ratio de explotación: el 85% de las veces será avaro, pero el 15% elige acciones
    dando lugar a la exploración.learning rate = se suele dejar en el 10 por cier
    dando lugar a las recompensas y permitiendo actualizar la importancia de cada
    Tras más iteraciones, mayor importancia tendrá esa acción.
discount_factor = También se suele empezar con valor de 0.1 pero aquí utilizamos
un valor del 0.2 para intentar indicar al algoritmo que nos interesa las recomper
```

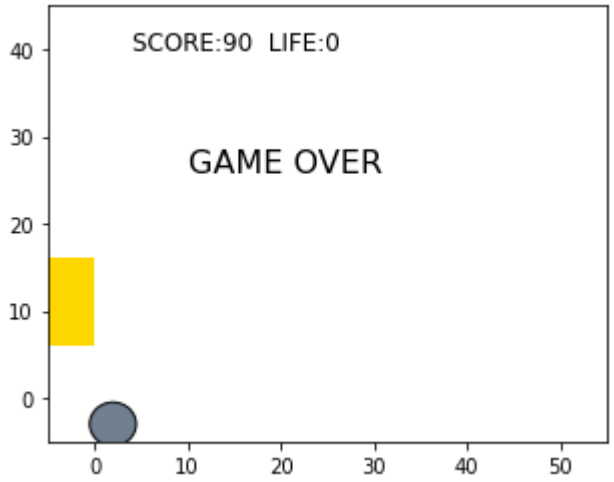
In [10]:
```
def play(rounds=5000, max_life=3, discount_factor = 0.1, learning_rate = 0.1,
         ratio_explotacion=0.9,learner=None, game=None, animate=False):

    if game is None:
        # si usamos movimiento_px = 5 creamos una tabla de politicas de 8x10
        # si usamos movimiento_px = 3 la tabla sera de 14x17
        game = PongEnvironment(max_life=max_life, movimiento_px = 3)

    if learner is None:
        print("Begin new Train!")
        learner = PongAgent(game, discount_factor = discount_factor,learning_rate

    max_points= -9999
    first_max_reached = 0
    total_rw=0
    steps=[]

    for played_games in range(0, rounds):
        state = game.reset()
        reward, done = None, None

        itera=0
        while (done != True) and (itera < 3000 and game.total_reward<=1000):
            old_state = np.array(state)
            next_action = learner.get_next_step(state, game)
            state, reward, done = game.step(next_action, animate=animate)
            if rounds > 1:
                learner.update(game, old_state, next_action, reward, state, done)
            itera+=1

        steps.append(itera)

        total_rw+=game.total_reward
        if game.total_reward > max_points:
            max_points=game.total_reward
            first_max_reached = played_games

        if played_games %500==0 and played_games >1 and not animate:
            print("-- Partidas[", played_games, "] Avg.Puntos[", int(total_rw/pla

    if played_games>1:
        print('Partidas[',played_games,'] Avg.Puntos[',int(total_rw/played_games)

    #learner.print_policy()

    return learner, game
```

In [ ]:
```
#Y vemos la salida del entreno, luego de unos 2 minutos:
```

In [11]:
```
learner, game = play(rounds=5000, discount_factor = 0.2, learning_rate = 0.1, rat
```

```
Begin new Train!
-- Partidas[ 500 ] Avg.Puntos[ 12 ]  AVG Steps[ 214 ] Max Score[ 130 ]
-- Partidas[ 1000 ] Avg.Puntos[ 16 ]  AVG Steps[ 225 ] Max Score[ 150 ]
-- Partidas[ 1500 ] Avg.Puntos[ 16 ]  AVG Steps[ 227 ] Max Score[ 150 ]
-- Partidas[ 2000 ] Avg.Puntos[ 20 ]  AVG Steps[ 240 ] Max Score[ 220 ]
-- Partidas[ 2500 ] Avg.Puntos[ 25 ]  AVG Steps[ 258 ] Max Score[ 260 ]
-- Partidas[ 3000 ] Avg.Puntos[ 28 ]  AVG Steps[ 268 ] Max Score[ 330 ]
-- Partidas[ 3500 ] Avg.Puntos[ 30 ]  AVG Steps[ 274 ] Max Score[ 330 ]
-- Partidas[ 4000 ] Avg.Puntos[ 32 ]  AVG Steps[ 279 ] Max Score[ 330 ]
-- Partidas[ 4500 ] Avg.Puntos[ 34 ]  AVG Steps[ 287 ] Max Score[ 390 ]
Partidas[ 4999 ] Avg.Puntos[ 36 ] Max score[ 390 ] en partida[ 4462 ]
```

In [16]:
```
learner2 = PongAgent(game, policy=learner.get_policy())
learner2.ratio_explotacion = 1.0  # con esto quitamos las elecciones aleatorias d
player = play(rounds=1, learner=learner2, game=game, animate=True)
```



In [14]:
```
#En las salidas vemos sobre todo cómo va mejorando en la cantidad de “steps”
#que da el agente antes de perder la partida.
```

In [15]:
```
#Ya contamos con nuestro agente entrenado, ahora veamos qué tal se comporta en ur
#y lo podemos ver jugar, pasando el parámetro animate=True.

#Antes de jugar, instanciamos un nuevo agente
#“learner2” que utilizará las políticas que creamos anteriormente.
#A este agente le seteamos el valor de explotación en 1, para evitar que tome pas
```

In [ ]:
```
#rESULTADO
```

In [1]:
```
from neo4j import GraphDatabase
from neomodel import (config, StructuredNode, StringProperty, IntegerProperty,
    UniqueIdProperty, RelationshipTo, RelationshipFrom)
import tweepy
import json, csv, sys
```

In [2]:
```
graphdb=GraphDatabase.driver(uri="bolt://localhost:7687", auth=("neo4j", "cuenca"
```

In [3]:
```
Consumer_Key = 'jb2YpS4N4VirQ9AP7QvfuSkwS'
Consumer_Key_Secret = '1Pww9wmuxNrmgzmuUoD7XFG1ZLBEQxlFLodFiaqlzgG4ilfur7'
Access_Token = '1173603361339445249-xsCW5e9Y6vCdvfyz9uRlAUEvdsRJbX'
Access_Token_Secret = 'oZ98Xco7cfc7EIeql4Bq3tSoUGlRhXzZSssRMEebhwtYO'

auth = tweepy.OAuthHandler(Consumer_Key, Consumer_Key_Secret)
auth.set_access_token(Access_Token, Access_Token_Secret)

api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True)

dataMe = api.me()
```

In [4]:
```
config.DATABASE_URL = 'bolt://neo4j:cuenca@localhost:7687'

class Presidente(StructuredNode):
    namePresidente = StringProperty(unique_index=True)
    partidoPresidente = StringProperty(unique_index=True)

class Tweet(StructuredNode):
    idTweet = UniqueIdProperty()
    dateTweet = StringProperty(unique_index=False)
    messageTweet = StringProperty(unique_index=False)
    deviceTweet = StringProperty(unique_index=False)
    presidenteTweet = RelationshipTo('Presidente', 'Tweet_del_Presidente')
    usuarioTweet = RelationshipTo('Usuario', 'Direccion_del_Tweet')

class Usuario(StructuredNode):
    idUser = UniqueIdProperty()
    nameUser = StringProperty(unique_index=True)
    descriptionUser = StringProperty(unique_index=False)
    addressUser = StringProperty(unique_index=False)
    tweetsUser = RelationshipFrom('Tweet', 'Usuario_Tweet')
```

In [5]:
```
c = 0
presidenteT = Presidente(namePresidente="Guillermo Lasso", partidoPresidente="CRE
for tweet in tweepy.Cursor(api.search, q="Guillermo Lasso", tweet_mode = "extende
    '''Para mostrar todos los datos'''
    print("************************"+str(c))
    item = json.loads(json.dumps(tweet._json, indent=3))
    '''Datos de tweet '''
    iditem = item['id']
    dateTweet = item['created_at']
    message = item['full_text']
    dipositivo = item['source']
    print("Fecha Tweet "+ dateTweet)
    print("Id Tweet "+ str(iditem))
    print("Mesaje" + message)
    print("Dispositivo "+ dipositivo)

    '''Datos de usuario'''
    user = item['user']
    useid = user['id']
    usename = user['name']
    usedescription = user['description']
    useaddress = user['location']
    print("Usuario ID: " + str(useid))
    print("Nombre de Usuario: " + usename)
    print("Descripcion del Mensaje: " + usedescription)
    print("Direccion " + useaddress)

    tweet = Tweet(idTweet=iditem, dateTweet=dateTweet, messageTweet=message, devi
    userT =  Usuario(idUser=useid, nameUser=usename, descriptionUser=usedescripti
    tweet.usuarioTweet.connect(userT)
    tweet.presidenteTweet.connect(presidenteT)
    print("-----------------------------------")

    c=c+1
```

```
************************0
Fecha Tweet Thu Jun 10 00:32:29 +0000 2021
Id Tweet 1402785707714830342
MesajeRT @DDavidVillamar: "En el triunfo del señor Guillermo Lasso, Radio Vis
ión jugó un papel fundamental. No se olviden. NO SE OLVIDEN..."
Pala…
Dispositivo <a href="https://mobile.twitter.com" rel="nofollow">Twitter Web A
pp</a>
Usuario ID: 175450384
Nombre de Usuario: ®©™
Descripcion del Mensaje: No busco quién me entienda sino alguíen que me escuc
he.. No escribo para los demás, escribo para mí.
Direccion
-----------------------------------
************************1
Fecha Tweet Thu Jun 10 00:31:56 +0000 2021
Id Tweet 1402785569604833293
MesajeRT @DDavidVillamar: "En el triunfo del señor Guillermo Lasso, Radio Vis
ión jugó un papel fundamental. No se olviden. NO SE OLVIDEN..."
Pala
```