



Universidad Veracruzana

Universidad Veracruzana

Región Xalapa

Instituto de Investigaciones en Inteligencia Artificial

Uso de GitHub Copilot para la programación de OpenAI Gym

Presenta:
Terrones Pacheco Alex Antonio

Director:
Dr. Fernando Martín Montes González

Diciembre de 2024

“Lis de Veracruz: Arte, Ciencia, Luz”



Tabla de contenido

<i>Introducción</i>	4
<i>¿Qué es OpenAI Gym?</i>	5
<i>Definición del Proyecto:</i>	9
<i>Implementación del Proyecto y Diseño de Agentes</i>	11
<i>Agentes generados con Copilot</i>	18
Primer agente diseñado con Copilot	19
Segundo agente diseñado con Copilot	26
Tercer agente diseñado con Copilot.....	31
<i>OpenAI Gym con ROS en Rviz</i>	35
<i>Agentes entrenados para visualización en RViz</i>	51
<i>Conclusión</i>	59
<i>Contexto y Consideraciones Éticas en el Uso de Inteligencia Artificial Generativa en la Educación</i>	62
<i>Referencias</i>	64
Agente 1: CartPole de prueba en ejecución	12
Agente 2: Acrobot en ejecución	23
Agente 3: Agente MountainCar en ejecución	30
Agente 4: Agente Pendulum en ejecución	34
Código 1: Código de la primer prueba de CartPole	12
Código 2: Código del Agente completo de CartPole	13
Código 3: Código del Agente Acrobot	19
Código 4: Código del agente Acrobot mejorado	21
Código 5: Código del agente completo de Acrobot	22
Código 6: Código del agente Acrobot optimizado	24
Código 7: Código de MountainCar	26
Código 8: Código con parte grafica MountainCar	27
Código 9: Código Final MountainCar	28
Código 10: Primer código del agente pendulum	31

Código 11: Código Final de agente Pendulum.....	32
Código 12: gym_agent_node.py.....	38
Código 13: Acrobot_agent.py	42
Código 14: Mountaintcar_agent.py.....	45
Código 15: Pendulum_Agent.py.....	48
Código 16: Código Acrobot_trained	52
Código 17: Código Mountaintcar_trained.....	54
Gráfica 1: Recompensas del agente CartPole en una ejecución de 1000 episodios	14
Gráfica 2: Grafica de recompensas Acrobot.....	25
Gráfica 3: Grafica de recompensas MountainCar	29
Gráfica 4:Grafica de recompensas Pendulum	33
FOTO 1: Recompensas impresas en la consola [Gráfica 1].....	14
FOTO: 2: Errores del Agente Acrobot.....	20
FOTO 3: Recompensas en consola agente MountainCar	29
FOTO 4: Recompensas en consola agente MountainCar	30
FOTO 5: Recompensas en pantalla agente Pendulum.....	33
FOTO 6: Orden de los archivos	37
FOTO 7: Cart Pole en Rviz	39
FOTO 8: Acrobot en Rviz.....	44
FOTO 9: Mountain Car en Rviz	47
FOTO 10:Pendulum en Rviz	50
FOTO 11: Recompensas en consola Acrobot_trained	51
FOTO 12: Recompensas en consola Mountaintcar_trained.....	53

Introducción

En el contexto de la rápida evolución de la tecnología y el aprendizaje automático, los generadores de código automáticos como GitHub Copilot han surgido como herramientas revolucionarias para la programación. Estas herramientas están diseñadas para asistir a los desarrolladores, especialmente a aquellos que recién comienzan su trayectoria en la programación, facilitando la escritura de código y sugiriendo soluciones en tiempo real. El uso de GitHub Copilot en entornos educativos plantea preguntas cruciales sobre su impacto en el proceso de aprendizaje, particularmente en cómo los estudiantes principiantes pueden depender de esta tecnología para resolver tareas de programación complejas.

Este proyecto se centra en explorar el potencial de GitHub Copilot para programar agentes en OpenAI Gym, un entorno de simulación ampliamente utilizado en la investigación de aprendizaje por refuerzo. La premisa es investigar hasta qué punto Copilot puede generar código efectivo y cómo este proceso afecta el entendimiento de los estudiantes sobre los fundamentos de la programación. OpenAI Gym presenta un conjunto de desafíos específicos que requieren que los agentes aprendan a tomar decisiones basadas en recompensas y penalizaciones, lo que lo convierte en un campo ideal para probar las capacidades de un generador de código automático.

El objetivo de este proyecto radica en evaluar la eficacia de GitHub Copilot como una herramienta educativa. Se busca determinar si los estudiantes pueden confiar en Copilot para resolver problemas de programación sin comprometer su aprendizaje de conceptos fundamentales. La capacidad de Copilot para proporcionar soluciones precisas en lenguajes de programación como Python, JavaScript y C++ será evaluada en el contexto de OpenAI Gym, analizando si las sugerencias automáticas permiten a los estudiantes alcanzar una comprensión más profunda o, por el contrario, si promueven una dependencia que podría inhibir su desarrollo como programadores competentes.

Este estudio tiene el potencial de proporcionar información valiosa sobre las oportunidades y desafíos de incorporar generadores de código automáticos en la educación en programación. Los resultados obtenidos podrían influir en la manera en que se enseñan las habilidades de codificación en el futuro, promoviendo un enfoque equilibrado que combine

el poder de la inteligencia artificial con la necesidad de una sólida base conceptual en programación.

¿Qué es OpenAI Gym?

OpenAI Gym es una plataforma que proporciona una colección de entornos simulados donde los desarrolladores e investigadores pueden entrenar, probar y comparar agentes de inteligencia artificial, especialmente aquellos que utilizan algoritmos de aprendizaje por refuerzo (RL, por sus siglas en inglés). Fue lanzada por **OpenAI en 2016** y se ha convertido en un estándar en la investigación de aprendizaje por refuerzo debido a su facilidad de uso y la amplia gama de entornos disponibles.

Características Principales:

1. Diversidad de Entornos:

- OpenAI Gym ofrece una variedad de entornos que van desde simples juegos clásicos como el Atari, hasta simulaciones más complejas como la manipulación de robots. Esto permite a los desarrolladores probar sus algoritmos en diferentes escenarios.

Algunos entornos de OpenAI Gym:

- **CartPole-v1**
 - **Descripción:** Un carrito debe mantener un poste equilibrado sobre él moviéndose hacia la izquierda o la derecha.
 - **Objetivo:** Evitar que el poste caiga manteniéndolo en equilibrio el mayor tiempo posible.
- **MountainCar-v0**
 - **Descripción:** Un coche en un valle debe llegar a la cima de una montaña, pero su motor no es lo suficientemente fuerte para subir directamente.
 - **Objetivo:** Oscilar hacia adelante y hacia atrás para ganar impulso y alcanzar la cima.

- **Acrobot-v1**
 - **Descripción:** Un sistema de dos segmentos conectados por articulaciones debe balancearse para alcanzar una meta en la parte superior.
 - **Objetivo:** Elevar la parte superior del "brazo" por encima de una línea predefinida.
- **Pendulum-v1**
 - **Descripción:** Un péndulo invertido debe mantenerse lo más cerca posible de la posición vertical.
 - **Objetivo:** Balancear el péndulo hasta que esté completamente vertical.
- **LunarLander-v2**
 - **Descripción:** Un módulo lunar debe aterrizar de forma segura controlando sus propulsores.
 - **Objetivo:** Aterrizar suavemente en la zona objetivo.
- **BipedalWalker-v3**
 - **Descripción:** Un robot bípedo debe aprender a caminar sobre terreno accidentado.
 - **Objetivo:** Avanzar lo más lejos posible sin caerse.
- **FrozenLake-v1**
 - **Descripción:** Un agente debe navegar en una cuadrícula resbaladiza, evitando caer en agujeros.
 - **Objetivo:** Llegar al punto objetivo sin caer en los agujeros.
- **Taxi-v3**
 - **Descripción:** Un taxi debe recoger pasajeros y llevarlos a su destino en una cuadrícula.
 - **Objetivo:** Llevar al pasajero a su destino en el menor número de movimientos.
- **Atari Environments (Pong, Breakout, Space Invaders, etc.)**

- **Descripción:** Juegos clásicos de Atari, como Pong o Breakout.
- **Objetivo:** Maximizar la puntuación según las reglas del juego específico.
- **Reacher-v2**
 - **Descripción:** Un brazo robótico debe alcanzar un punto objetivo en un espacio bidimensional.
 - **Objetivo:** Tocar el punto objetivo con el extremo del brazo.
- **Ant-v4**
 - **Descripción:** Un robot cuadrúpedo debe aprender a caminar en un terreno plano.
 - **Objetivo:** Avanzar lo más lejos posible manteniendo la estabilidad.
- **Humanoid-v4**
 - **Descripción:** Un robot humanoide debe aprender a caminar y mantener el equilibrio.
 - **Objetivo:** Caminar lo más lejos posible sin caerse.

En general, hay más de **100 entornos** disponibles en OpenAI Gym si contamos todas las versiones (v0, v1, etc.) y variaciones dentro de cada categoría, especialmente en juegos de Atari y los entornos de simulación de robótica como Ant y Humanoid.

2. Interfaz Estándar:

- Gym proporciona una interfaz estándar para interactuar con los entornos. Esto incluye funciones para restablecer el entorno, realizar acciones y recibir observaciones y recompensas. Esta interfaz estandarizada facilita el desarrollo de agentes que pueden ser fácilmente trasladados de un entorno a otro.

3. Aprendizaje por Refuerzo:

- OpenAI Gym está diseñado específicamente para el aprendizaje por refuerzo, una rama de la inteligencia artificial en la que un agente aprende a tomar decisiones a través de la interacción con un entorno, optimizando una recompensa a largo plazo.

4. Integración con Herramientas de IA:

- OpenAI Gym se puede integrar fácilmente con otras bibliotecas de Python, como TensorFlow, PyTorch, y Keras, que son comúnmente utilizadas para construir y entrenar modelos de aprendizaje profundo.

Aplicaciones de OpenAI Gym:

- **Investigación Académica:** OpenAI Gym es ampliamente utilizado en la investigación académica para experimentar con nuevos algoritmos de aprendizaje por refuerzo y compararlos con métodos existentes.
- **Desarrollo de Agentes de IA:** Los desarrolladores lo utilizan para crear agentes que pueden aprender a realizar tareas complejas, como jugar videojuegos o controlar robots.
- **Educación:** Es una herramienta valiosa para enseñar conceptos de aprendizaje por refuerzo, ya que proporciona un entorno práctico y accesible para experimentar.

Definición del Proyecto:

El presente proyecto tiene como objetivo explorar el potencial de GitHub Copilot para la programación de agentes en OpenAI Gym. GitHub Copilot es una herramienta de generación automática de código basada en inteligencia artificial desarrollada por GitHub. Su propósito es asistir a los desarrolladores proporcionando sugerencias en tiempo real mientras escriben código, facilitando así el proceso de programación, especialmente para principiantes. OpenAI Gym, por su parte, es una plataforma de simulación diseñada para el desarrollo y la evaluación de algoritmos de aprendizaje por refuerzo.

El proyecto se centrará en determinar la eficacia de GitHub Copilot al generar código para agentes que operan dentro del entorno de OpenAI Gym. En particular, se investigará si Copilot puede generar código funcional que permita a los agentes realizar tareas específicas dentro de estos entornos simulados. Este análisis se llevará a cabo sin la comparación directa del código generado con el escrito manualmente; en su lugar, se evaluará exclusivamente la funcionalidad del código sugerido por Copilot.

Metodología:

Configuración del Entorno:

Instalación de GitHub Copilot: Asegurarse de tener GitHub Copilot habilitado en un editor de código (por ejemplo, Visual Studio Code). Esto permitirá utilizar la herramienta para generar código automáticamente basado en las descripciones y comentarios que se proporcionen.

Preparación de OpenAI Gym: Confirmar que OpenAI Gym está correctamente instalado y configurado en su entorno de desarrollo. Asegurarse de poder crear y ejecutar entornos de simulación, como por ejemplo "CartPole-v1".

Desarrollo del Agente:

Especificación de Tareas: Definir las tareas que el agente debe realizar en el entorno de OpenAI Gym. Por ejemplo, en el entorno "CartPole-v1", el objetivo es mantener el equilibrio del poste durante el mayor tiempo posible.

Generación de Código con GitHub Copilot: Utilizar GitHub Copilot para escribir el código necesario para que el agente realice las tareas especificadas. Proporcionar descripciones detalladas y comentarios en tu código para guiar a Copilot en la generación de sugerencias.

Evaluación de la Funcionalidad:

Pruebas Iniciales: Ejecutar el código generado por Copilot para verificar si el agente puede realizar las tareas en el entorno de OpenAI Gym. Observar si el agente cumple con los objetivos establecidos, como mantener el equilibrio en "CartPole-v1".

Monitoreo de Resultados: Durante las pruebas, monitorear el rendimiento del agente y registrar cualquier comportamiento inusual o errores. Asegurarse de que el código sea funcional y que el agente actúe según lo esperado en el entorno.

Implementación del Proyecto y Diseño de Agentes

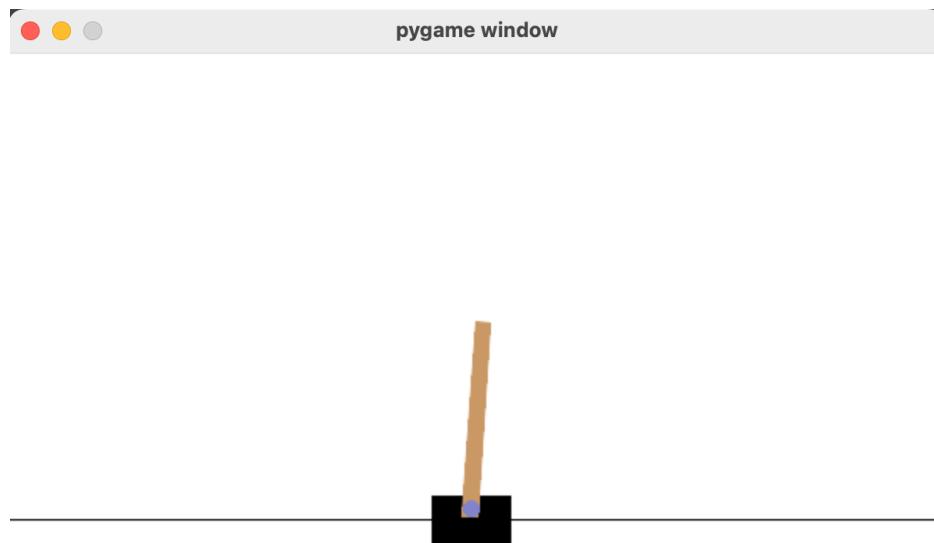
Preparación del Entorno de Trabajo:

OpenAI Gym se instaló en un dispositivo con sistema operativo macOS, con ayuda de homebrew se creó un entorno específico con miniconda y su comando “conda” solo para este proyecto en el que se instalaron todas las dependencias necesarias como: pip, python, matplotlib con pip se instaló gym y gymnasium con “pip install gym”, “pip install gymnasium” gymnasium fue instalado para probar que estuviera funcionando bien y no hubiera problemas de versiones.

Para el uso de Github Copilot en Visual Studio Code se instaló la extensión de “Github Copilot” y “Github Copilot Chat”. Se creó un primer código de forma manual para verificar que todo estuviera funcionando de manera adecuada y se le agregó un bloque try para que la simulación se quedara en ejecución hasta que se termine el proceso de forma manual.

```
● ● ●  
1 import gym  
2 import time  
3  
4 env = gym.make('CartPole-v1', render_mode='human')  
5 env.reset()  
6  
7 try:  
8     for _ in range(1000):  
9         action = env.action_space.sample()  
10        observation, reward, terminated, truncated, info = env.step(action)  
11        if terminated or truncated:  
12            env.reset()  
13  
14        env.render() # Renderizar el entorno en cada iteración  
15  
16    except KeyboardInterrupt:  
17        print("Cerrando el entorno...")  
18  
19    # Cerrar el entorno  
20    env.close()  
21  
22
```

Código 1: Código de la primera prueba de CartPole



Agente 1: CartPole de prueba en ejecución

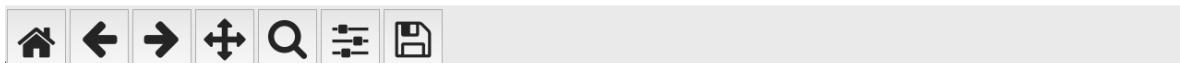
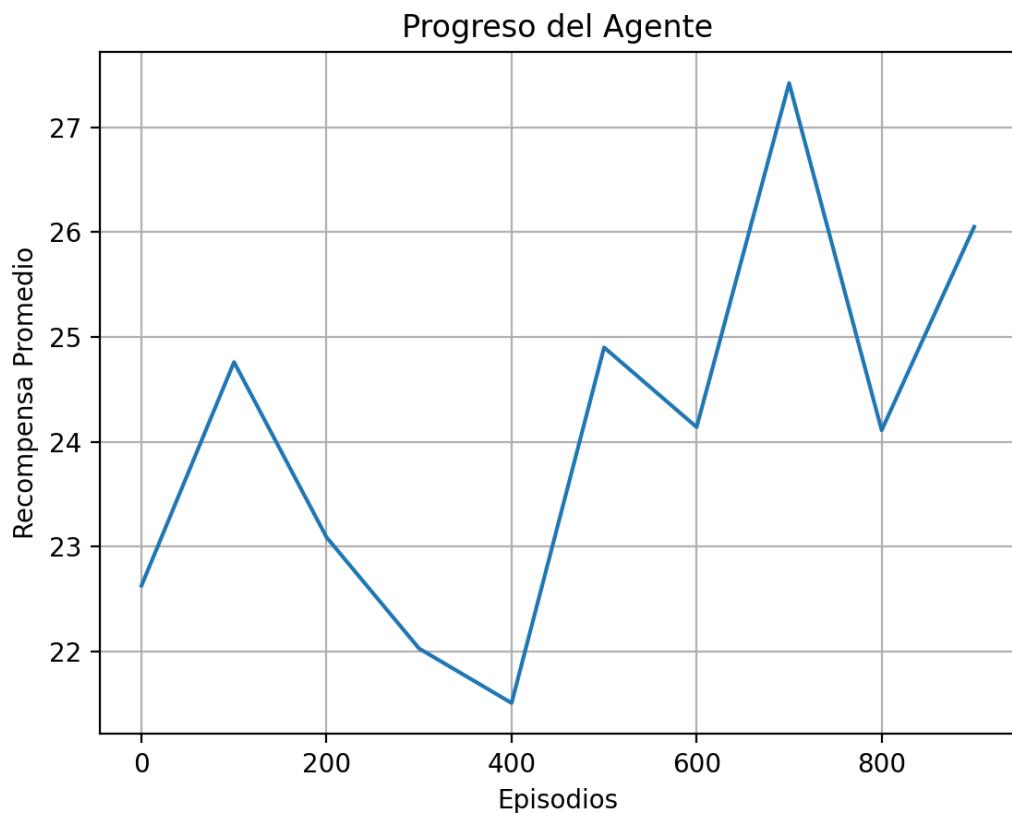
Despues se creó un segundo código en el que se recopilaron los datos de la simulación con ayuda de *matplotlib*.

```

● ● ●
1 import gym
2 Import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Crear el entorno
6 env = gym.make('CartPole-v1', render_mode='human')
7 state_space = (24, 24, 24, 24) # Ajustar el espacio de estados discretizado
8 action_space = env.action_space.n
9
10 # Verificar el tipo de estado retornado por env.reset()
11 state = env.reset()
12
13 # Parámetros de Q-learning
14 learning_rate = 0.1
15 discount_factor = 0.99
16 epsilon = 1.0
17 epsilon_decay = 0.995
18 epsilon_min = 0.01
19
20 # Inicialización de la tabla Q
21 q_table = np.zeros(state_space + (action_space,))
22
23 # Función para discretizar el espacio continuo de estados
24 def discretize_state(state):
25     if isinstance(state, tuple):
26         state_array, _ = state # Extraer el array del estado
27     else:
28         state_array = state # Estado ya es un array
29
30     bins = [np.linspace(-2.4, 2.4, num=state_space[0]),
31             np.linspace(-2.0, 2.0, num=state_space[1]),
32             np.linspace(-0.5, 0.5, num=state_space[2]),
33             np.linspace(-0.5, 0.5, num=state_space[3])]
34
35     # Discretizar cada dimensión del estado
36     state_indices = [np.digitize(state_array[i], bins[i]) - 1 for i in range(len(state_array))]
37
38     # Asegurarse de que los índices estén dentro del rango permitido
39     state_indices = np.clip(state_indices, 0, np.array(state_space) - 1)
40
41     return tuple(state_indices)
42
43
44 # Listas para monitoreo
45 episode_rewards = []
46
47 # Entrenamiento del agente
48 for episode in range(1000):
49     state = discretize_state(env.reset())
50     total_reward = 0
51     done = False
52
53     while not done:
54         if np.random.rand() < epsilon:
55             action = env.action_space.sample()
56         else:
57             action = np.argmax(q_table[state])
58
59         next_state, reward, terminated, truncated, info = env.step(action)
60         # print(f"Next State: {next_state}") # Imprimir next_state
61         next_state = discretize_state(next_state)
62
63         best_next_action = np.argmax(q_table[next_state])
64         q_table[state][action] += learning_rate * (reward + discount_factor * q_table[next_state][best_next_action] - q_table[state][action])
65     state = next_state
66     total_reward += reward
67
68     if terminated or truncated:
69         break
70
71     # Reducir epsilon
72     epsilon = max(epsilon_min, epsilon * epsilon_decay)
73
74     # Guardar las recompensas
75     episode_rewards.append(total_reward)
76
77     if episode % 100 == 0:
78         avg_reward = np.mean(episode_rewards[-100:])
79         print(f"Episodio: {episode}, Recompensa Total: {total_reward}, Recompensa Promedio: {avg_reward}")
80
81     # Cerrar el entorno
82     env.close()
83
84     # Monitoreo y visualización
85     # Calcular recompensas promedio cada 100 episodios
86     episodes = list(range(0, 1000, 100))
87     average_rewards = [np.mean(episode_rewards[i:i + 100]) for i in range(0, len(episode_rewards), 100)]
88
89     # Graficar el progreso
90     plt.plot(episodes, average_rewards)
91     plt.xlabel('Episodios')
92     plt.ylabel('Recompensa Promedio')
93     plt.title('Progreso del Agente')
94     plt.grid(True)
95     plt.savefig('agent_progress.png')
96     plt.show()
97
98     # Guardar la tabla Q
99     np.save('q_table.npy', q_table)
100
101

```

Código 2: Código del Agente completo de CartPole



Gráfica 1: Recompensas del agente CartPole en una ejecución de 1000 episodios

```

> python /Users/alex/Desktop/openaigym2.py
2024-09-12 13:12:10.901 python[81391:616360] +[IMKClient subclass]: chose IMKClient_Legacy
2024-09-12 13:12:10.901 python[81391:616360] +[IMKInputSession subclass]: chose IMKInputSession_Legacy
Episodio: 0, Recompensa Total: 26.0, Recompensa Promedio: 26.0
Episodio: 100, Recompensa Total: 23.0, Recompensa Promedio: 22.6
Episodio: 200, Recompensa Total: 17.0, Recompensa Promedio: 24.7
Episodio: 300, Recompensa Total: 15.0, Recompensa Promedio: 23.07
Episodio: 400, Recompensa Total: 13.0, Recompensa Promedio: 22.01
Episodio: 500, Recompensa Total: 25.0, Recompensa Promedio: 21.63
Episodio: 600, Recompensa Total: 25.0, Recompensa Promedio: 24.9
Episodio: 700, Recompensa Total: 32.0, Recompensa Promedio: 24.21
Episodio: 800, Recompensa Total: 18.0, Recompensa Promedio: 27.28
Episodio: 900, Recompensa Total: 56.0, Recompensa Promedio: 24.49

```

FOTO 1: Recompensas impresas en la consola [Gráfica 1]

El tipo de agente que usado en el código es un **agente de aprendizaje por refuerzo basado en Q-learning**. Específicamente, es un agente que utiliza una **tabla Q** (Q-table) para aprender una política óptima en un entorno discreto, en este caso, el entorno *CartPole-v1* de OpenAI Gym.

Características del agente:

1. **Q-learning:** Es un algoritmo de aprendizaje por refuerzo que busca aprender la mejor acción a tomar en cada estado, actualizando los valores de la *tabla Q* según las recompensas obtenidas.
2. **Exploración y explotación:** El agente utiliza una política **ϵ -greedy**, donde explora tomando acciones aleatorias con probabilidad ϵ y explota las mejores acciones conocidas (según la tabla Q) el resto del tiempo.
3. **Espacio de acciones:** Se define por el número de acciones posibles que el agente puede realizar en el entorno, determinado por *env.action_space.n*.
4. **Discretización del estado:** Dado que el estado del entorno *CartPole-v1* es continuo, el código discretiza el espacio de estados utilizando bins para poder manejar la tabla Q.

Este tipo de agente es adecuado para entornos donde el espacio de estados puede ser discretizado y el número de acciones es manejable, como es el caso de *CartPole-v1*.

La **Q-table** es una parte fundamental del algoritmo de **Q-learning**, es una técnica de **aprendizaje por refuerzo** que utiliza esta tabla para almacenar las **recompensas esperadas** para cada combinación de estado y acción.

¿Qué es la Q-table?

- La **Q-table** es una tabla que mapea cada par (estado, acción) a un valor conocido como **valor Q**. Este valor representa la **recompensa esperada** a largo plazo que el agente puede obtener al tomar una acción específica en un estado particular, y luego seguir la política aprendida.

- La tabla tiene una estructura bidimensional, donde las filas representan los **estados** y las columnas representan las **acciones**. Cada celda de la tabla contiene el valor Q para una acción en un estado dado.

¿Cómo se actualiza la Q-table en Q-learning?

El agente actualiza la Q-table después de cada interacción con el entorno usando la siguiente fórmula de actualización:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

Donde:

- $Q(s, a)$ es el valor Q actual para el estado s y la acción a .
- α es la **tasa de aprendizaje**, que determina cuánto influye el nuevo conocimiento en el valor Q.
- r es la **recompensa** recibida después de tomar la acción a en el estado s .
- γ es el **factor de descuento**, que determina la importancia de las recompensas futuras.
- $\max_{a'} Q(s', a')$ es el valor máximo Q para el siguiente estado s' , evaluado sobre todas las acciones a' .

El agente utiliza una **Q-table** para aprender qué acciones tomar en cada estado del entorno CartPole-v1, basándose en la experiencia acumulada mediante las recompensas obtenidas. La tabla Q es la estructura que almacena este conocimiento y se actualiza en cada paso del episodio usando la fórmula de Q-learning.

Razones por las que se seleccionó Q-learning:

1. **Simplicidad:** Q-learning es un algoritmo relativamente simple de implementar en comparación con otros métodos más avanzados como Deep Q-Networks (DQN). No requiere redes neuronales, lo que simplifica la estructura del agente.

2. **Entornos discretos:** El entorno CartPole-v1 tiene un espacio de acciones discretas (moverse a la izquierda o a la derecha), lo que se adapta bien al uso de una tabla Q para mapear estados y acciones.
3. **Estabilidad en entornos pequeños:** Dado que el espacio de estados del CartPole-v1 puede ser discretizado en un número manejable de estados, Q-learning es eficiente en términos de almacenamiento y tiempo de computación.

Para probar el agente en el entorno de OpenAI Gym, se comenzó configurando los parámetros iniciales del algoritmo de Q-learning. El parámetro “learning_rate” controla cuánto afecta una nueva recompensa en la tabla Q, mientras que “discount_factor” establece la importancia de las recompensas futuras. “Epsilon” se usa para balancear la exploración y explotación durante el entrenamiento. Utilicé una tasa de aprendizaje (`learning_rate`) de 0.1, un factor de descuento (`discount_factor`) de 0.99, y un valor inicial de epsilon de 1.0 para permitir exploración al inicio, con un decaimiento progresivo (`epsilon_decay`) de 0.995 para disminuir la exploración conforme avanzaba el entrenamiento.

Inicialmente, los resultados mostraron que el agente alcanzaba una recompensa de 26 en el primer episodio. A lo largo de las primeras 100 iteraciones, el rendimiento no mejoraba significativamente, con recompensas promedio de 22.6 y 24.7, lo que indicaba que el agente aún no había aprendido una política eficiente.

Durante las pruebas, se observó que la recompensa fluctuaba y no mostraba mejoras consistentes pero se deben ajustar lentamente los parámetros como la exploración y permitir más explotación de las mejores acciones aprendidas, para comprobar si el rendimiento mejora.

Este comportamiento sugeriría que el agente estaba aprendiendo, pero que aún necesitaba más ajustes para optimizar su rendimiento, como el ajuste fino de la tasa de aprendizaje o la ampliación de la tabla Q.

Agentes generados con Copilot

Para comenzar con el diseño de agentes con Copilot se seguirán los siguientes puntos:

Diseño del Agente:

- **Utilizar GitHub Copilot:** Utilizar GitHub Copilot para generar el código del agente. Como empezar con plantillas o fragmentos de código sugeridos por Copilot para crear la estructura básica de tu agente.
- **Implementar el Algoritmo:** Implementar el algoritmo de aprendizaje por refuerzo sugerido por Copilot.
- **Ejecutar Pruebas Iniciales:** Ejecutar pruebas iniciales para verificar que el agente funcione correctamente.
- **Ajustar Parámetros:** Ajustar los parámetros del agente (como la tasa de aprendizaje, el factor de descuento, y la tasa de exploración) basándose en los resultados obtenidos durante las pruebas.
- **Ejecutar Episodios:** Entrenar el agente en el entorno de OpenAI Gym durante un número determinado de episodios.
- **Monitorear el Rendimiento:** Registrar el rendimiento del agente (recompensas obtenidas, éxito en tareas, etc.).
- **Documentar el Proceso:** Documentar todo el proceso de desarrollo, desde la implementación del agente hasta los resultados obtenidos.

Primer agente diseñado con Copilot

Acrobot.

Solicitud: “Diseña un agente utilizando el entorno acrobot”

Código Generado:



```
1 import gym
2
3 try:
4     # Crear el entorno de Acrobot
5     env = gym.make('Acrobot-v1')
6
7     # Reiniciar el entorno
8     observation = env.reset()
9
10    for _ in range(1000):
11        env.render()
12
13        # Tomar una acción aleatoria
14        action = env.action_space.sample()
15
16        # Aplicar la acción en el entorno
17        observation, reward, done, info = env.step(action)
18
19        if done:
20            observation = env.reset()
21
22    finally:
23        env.close()
```

Código 3: Código del Agente Acrobot

En este código generado se encuentran algunos problemas que se muestran a continuación:

```
> /Users/alex/miniconda3/envs/gym_env/lib/python3.10/site-packages/gym/envs/classic_control/acrobot.py:281: UserWarning: WARN: You are calling render method without specifying any render mode. You can specify the render_mode at initialization, e.g. gym("Acrobot-v1", render_mode="rgb_array")
    logger.warn(
Traceback (most recent call last):
  File "/Users/alex/Desktop/acrobot.py", line 17, in <module>
    observation, reward, done, info = env.step(action)
ValueError: too many values to unpack (expected 4)
zsh: parse error near '\n'
```

FOTO: 2: Errores del Agente Acrobot

A lo que se le solicita a copilot solucionar los errores del código, y se le pide hacer que el brazo pueda llegar al objetivo, a lo que responde lo siguiente:

```

● ● ●

 1 import gym
 2 import numpy as np
 3 import random
 4
 5 # Parámetros de Q-learning
 6 alpha = 0.1 # Tasa de aprendizaje
 7 gamma = 0.99 # Factor de descuento
 8 epsilon = 0.1 # Probabilidad de exploración
 9 num_episodes = 1000 # Número de episodios de entrenamiento
10
11 # Crear el entorno de Acrobot
12 env = gym.make('Acrobot-v1')
13
14 # Inicializar la tabla Q
15 state_space_size = (1, 1, 1, 1, 1, 1) # Discretizar el espacio de estado
16 action_space_size = env.action_space.n
17 Q = np.zeros(state_space_size + (action_space_size,))
18
19 # Función para discretizar el estado
20 def discretize_state(state):
21     bins = np.array([10, 10, 10, 10, 10, 10])
22     discretized_state = tuple(((state + 1) * bins / 2).astype(int))
23     return discretized_state
24
25 # Entrenamiento
26 for episode in range(num_episodes):
27     state = discretize_state(env.reset())
28     done = False
29
30     while not done:
31         # Seleccionar una acción utilizando ε-greedy
32         if random.uniform(0, 1) < epsilon:
33             action = env.action_space.sample()
34         else:
35             action = np.argmax(Q[state])
36
37         # Ejecutar la acción
38         next_state, reward, done, truncated, info = env.step(action)
39         next_state = discretize_state(next_state)
40
41         # Actualizar la tabla Q
42         best_next_action = np.argmax(Q[next_state])
43         td_target = reward + gamma * Q[next_state][best_next_action]
44         td_error = td_target - Q[state][action]
45         Q[state][action] += alpha * td_error
46
47         state = next_state
48
49     if episode % 100 == 0:
50         print(f'Episode {episode}/{num_episodes}')
51
52     env.close()
53
54     # Guardar la tabla Q entrenada
55     np.save('q_table.npy', Q)

```

Código 4: Código del agente Acrobot mejorado

Pero aun cuenta con algunos errores el código, que provoca que no se ejecute correctamente el agente. Teniendo que solicitar exactamente 12 solicitudes para que

pudiera corregir los errores que impedían el correcto funcionamiento del agente, incluso se le tuvo que apoyar enviando el primer código completo [Código 2].

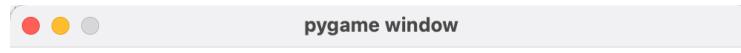
Resultando el código funcional de la siguiente manera:

```

● ● ●
1 import gym
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Crear el entorno
6 env = gym.make('Acrobot-v1', render_mode='human')
7 state_space = (10, 10, 10, 10, 10) # Ajustar el espacio de estados discretizado
8 action_space = env.action_space.n
9
10 # Parámetros de Q-learning
11 learning_rate = 0.1
12 discount_factor = 0.99
13 epsilon = 1.0
14 epsilon_decay = 0.995
15 epsilon_min = 0.01
16
17 # Inicialización de la tabla Q
18 q_table = np.zeros(state_space + (action_space,))
19
20 # Función para discretizar el espacio continuo de estados
21 def discretize_state(state):
22     if isinstance(state, dict):
23         state = state['observation']
24     state = np.array(state, dtype=np.float32) # Convertir el estado a un array de NumPy
25     bins = [np.linspace(-1, 1, num=state_space[i]) for i in range(len(state))]
26     state_indices = [np.digitize(state[i], bins[i]) - 1 for i in range(len(state))]
27     state_indices = np.clip(state_indices, 0, np.array(state_space) - 1)
28     return tuple(state_indices)
29
30 # Listas para monitoreo
31 episode_rewards = []
32
33 # Entrenamiento del agente
34 for episode in range(1000):
35     state = env.reset()
36     state = state[0] if isinstance(state, tuple) else state # Extraer el estado si es un tuple
37     state = state['observation'] if isinstance(state, dict) else state
38     state = discretize_state(state)
39     total_reward = 0
40     done = False
41
42     while not done:
43         if np.random.rand() < epsilon:
44             action = env.action_space.sample()
45         else:
46             action = np.argmax(q_table[state])
47
48         next_state, reward, terminated, truncated, info = env.step(action)
49         next_state = next_state[0] if isinstance(next_state, tuple) else next_state # Extraer el estado si es un tuple
50         next_state = next_state['observation'] if isinstance(next_state, dict) else next_state
51         next_state = discretize_state(next_state)
52
53         best_next_action = np.argmax(q_table[next_state])
54         q_table[state][action] += learning_rate * (reward + discount_factor * q_table[next_state][best_next_action] - q_table[state][action])
55         state = next_state
56         total_reward += reward
57
58         if terminated or truncated:
59             done = True
60
61     # Reducir epsilon
62     epsilon = max(epsilon_min, epsilon * epsilon_decay)
63
64     # Guardar las recompensas
65     episode_rewards.append(total_reward)
66
67     if episode % 100 == 0:
68         avg_reward = np.mean(episode_rewards[-100:])
69         print(f"Episodio: {episode}, Recompensa Total: {total_reward}, Recompensa Promedio: {avg_reward}")
70
71     # Cerrar el entorno
72     env.close()
73
74     # Monitoreo y visualización
75     # Calcular recompensas promedio cada 100 episodios
76     episodes = list(range(0, 1000, 100))
77     average_rewards = [np.mean(episode_rewards[i:i+100]) for i in range(0, len(episode_rewards), 100)]
78
79     # Graficar el progreso
80     plt.plot(episodes, average_rewards)
81     plt.xlabel('Episodios')
82     plt.ylabel('Recompensa Promedio')
83     plt.title('Progreso del Agente')
84     plt.grid(True)
85     plt.savefig('agent_progress.png')
86     plt.show()
87
88     # Guardar la tabla Q
89     np.save('q_table.npy', q_table)
90

```

Código 5: Código del agente completo de Acrobot



Agente 2: Acrobot en ejecución

Para acelerar el proceso de evaluación de los agentes se va a desactivar la Visualización durante el Entrenamiento:

- Se crea el entorno sin el modo de renderización (render_mode='human').
- Se crea un nuevo entorno con el modo de renderización activado (render_mode='human') solo para la evaluación.
- Se imprime información sobre el progreso del entrenamiento cada 100 episodios.

Estos cambios aceleran el proceso de entrenamiento y permitirte ver los resultados de la evaluación sin tener que esperar a que termine todo el entrenamiento ademas de aumentar los episodios de entrenamiento en menor tiempo.

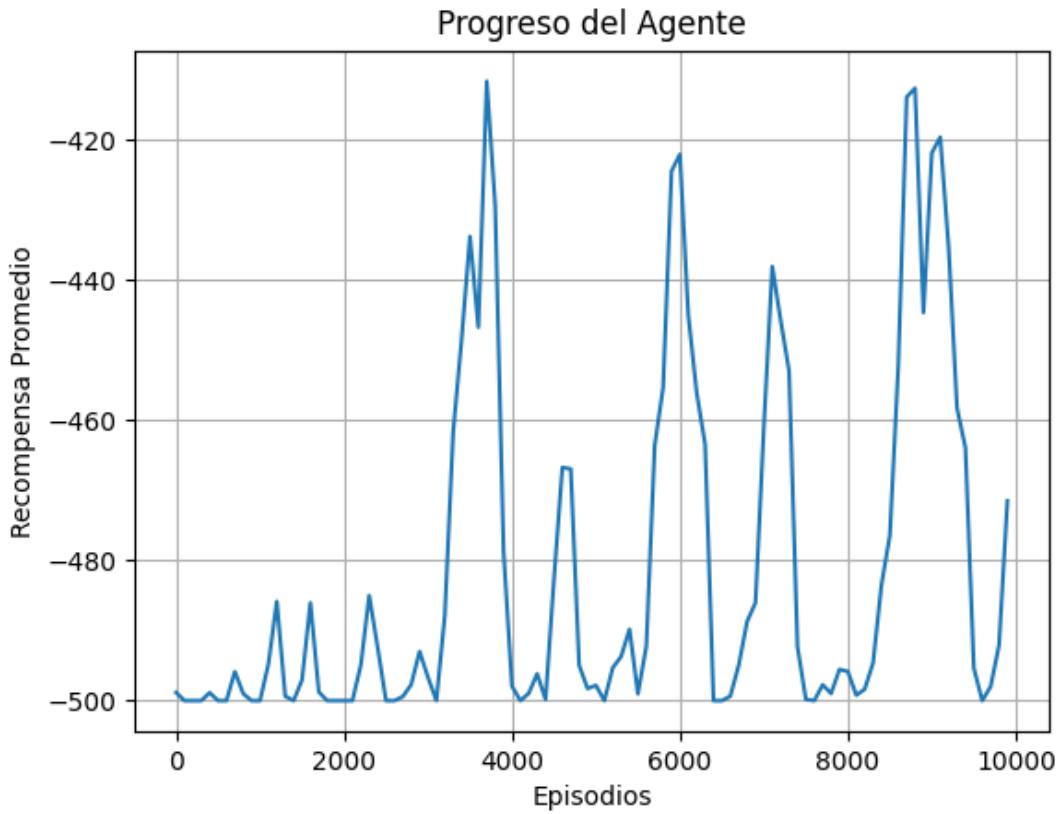
Quedando de la siguiente manera:

```

● ● ●
1 import gym
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Crear el entorno
6 env = gym.make('Acrobot-v1')
7 state_space = (12, 12, 12, 12, 12, 12) # Ajustar el espacio de estados discretizado
8 action_space = env.action_space.n
9
10 # Parámetros de Q-learning
11 learning_rate = 0.05 # Ajustar la tasa de aprendizaje
12 discount_factor = 0.98 # Ajustar el factor de descuento
13 epsilon = 1.0
14 epsilon_decay = 0.995
15 epsilon_min = 0.01
16
17 # Inicialización de la tabla Q
18 q_table = np.zeros(state_space + (action_space,))
19
20 # Función para discretizar el espacio continuo de estados
21 def discretize_state(state):
22     if isinstance(state, dict):
23         state = state['observation']
24     state = np.array(state, dtype=np.float32) # Convertir el estado a un array de NumPy
25     bins = np.linspace(-1, 1, num=state_space[i]) for i in range(len(state))
26     state_indices = [np.digitize(state[i], bins[i]) - 1 for i in range(len(state))]
27     state_indices = np.clip(state_indices, 0, np.array(state_space) - 1)
28     return tuple(state_indices)
29
30 # Listas para monitoreo
31 episode_rewards = []
32
33 # Entrenamiento del agente
34 for episode in range(10000):
35     state = env.reset()
36     state = state[0] if isinstance(state, tuple) else state # Extraer el estado si es un tuple
37     state = state['observation'] if isinstance(state, dict) else state
38     state = discretize_state(state)
39     total_reward = 0
40     done = False
41
42     while not done:
43         if np.random.rand() < epsilon:
44             action = env.action_space.sample()
45         else:
46             action = np.argmax(q_table[state])
47
48         next_state, reward, terminated, truncated, info = env.step(action)
49         next_state = next_state[0] if isinstance(next_state, tuple) else next_state # Extraer el estado si es un tuple
50         next_state = next_state['observation'] if isinstance(next_state, dict) else next_state
51         next_state = discretize_state(next_state)
52
53         best_next_action = np.argmax(q_table[next_state])
54         q_table[state][action] += learning_rate * (reward + discount_factor * q_table[next_state][best_next_action] - q_table[state][action])
55
56         state = next_state
57         total_reward += reward
58
59         if terminated or truncated:
60             done = True
61
62         # Reducir epsilon
63         epsilon = max(epsilon_min, epsilon * epsilon_decay)
64
65         # Guardar las recompensas
66         episode_rewards.append(total_reward)
67
68         if episode % 100 == 0:
69             avg_reward = np.mean(episode_rewards[-100:])
70             print(f"Episodio: {episode}, Recompensa Total: {total_reward}, Recompensa Promedio: {avg_reward}")
71
72 # Evaluación del agente
73 env = gym.make('Acrobot-v1', render_mode='human') # Activar la visualización
74 state = env.reset()
75 state = state[0] if isinstance(state, tuple) else state # Extraer el estado si es un tuple
76 state = state['observation'] if isinstance(state, dict) else state
77 state = discretize_state(state)
78 done = False
79 total_reward = 0
80
81 while not done:
82     action = np.argmax(q_table[state])
83     next_state, reward, terminated, truncated, info = env.step(action)
84     next_state = next_state[0] if isinstance(next_state, tuple) else next_state # Extraer el estado si es un tuple
85     next_state = next_state['observation'] if isinstance(next_state, dict) else next_state
86     next_state = discretize_state(next_state)
87
88     state = next_state
89     total_reward += reward
90
91     if terminated or truncated:
92         done = True
93
94     print(f"Recompensa Total en Evaluación: {total_reward}")
95
96     # Cerrar el entorno
97     env.close()
98
99     # Monitoreo y visualización
100    # Calcular recompensas promedio cada 100 episodios
101    episodes = list(range(0, 10000, 100))
102    average_rewards = [np.mean(episode_rewards[i:i+100]) for i in range(0, len(episode_rewards), 100)]
103
104    # Graficar el progreso
105    plt.plot(episodes, average_rewards)
106    plt.xlabel('Episodios')
107    plt.ylabel('Recompensa Promedio')
108    plt.title('Progreso del Agente')
109    plt.grid(True)
110    plt.savefig('agent_progress.png')
111    plt.show()
112
113    # Guardar la tabla Q
114    np.save('q_table.npy', q_table)

```

Código 6: Código del agente Acrobot optimizado



Gráfica 2: Grafica de recompensas Acrobot

Si bien el agente mostró un funcionamiento adecuado en las pruebas iniciales, aún se requieren realizar más experimentos y ajustes en las variables clave para optimizar su rendimiento y garantizar que logre cumplir con el objetivo propuesto. Los parámetros actuales, aunque permiten observar avances en el comportamiento del agente, no han sido completamente afinados para maximizar su desempeño. Es necesario realizar un análisis más profundo sobre las variantes utilizadas, como el ajuste de hiperparámetros, el tiempo de entrenamiento y la estrategia de exploración-explotación. Estas modificaciones permitirán no solo mejorar la eficiencia del agente, sino también su capacidad para adaptarse a situaciones más complejas en el entorno de prueba.

Los avances observados hasta el momento son indicativos de la capacidad de GitHub Copilot para asistir en la creación de soluciones de programación funcionales, lo que abre la puerta a futuras investigaciones o ajustes para optimizar su desempeño en escenarios más complejos.

Segundo agente diseñado con Copilot

MountainCar

Solicitud: “Diseña un entorno de MountainCar”

Código Generado:



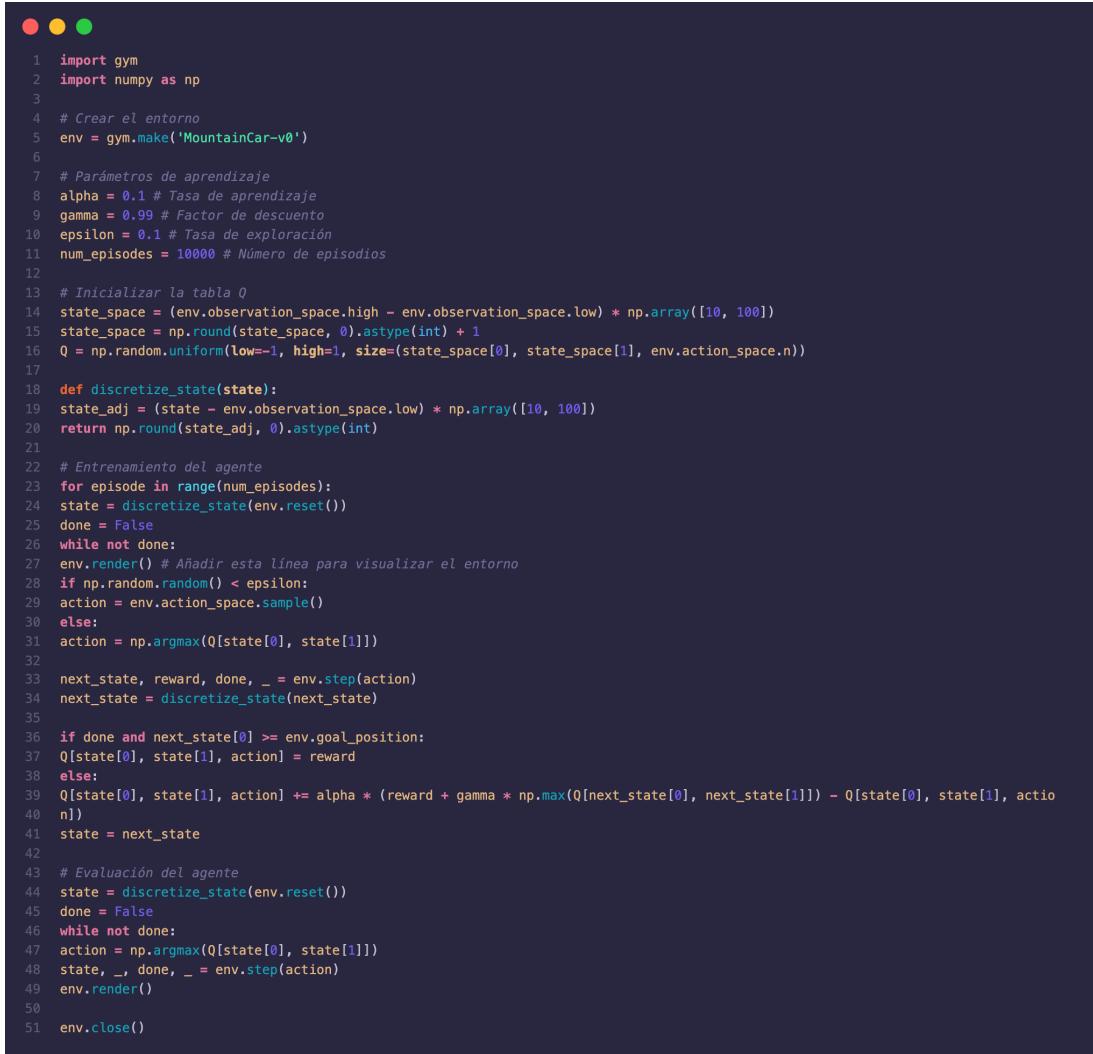
```
1 import gym
2 import numpy as np
3
4 # Crear el entorno
5 env = gym.make('MountainCar-v0')
6
7 # Parámetros de Q-learning
8 alpha = 0.1 # Tasa de aprendizaje
9 gamma = 0.99 # Factor de descuento
10 epsilon = 0.1 # Tasa de exploración
11 num_episodes = 10000 # Número de episodios
12
13 # Inicializar la tabla Q
14 state_space = (env.observation_space.high - env.observation_space.low) * np.array([10, 100])
15 state_space = np.round(state_space, 0).astype(int) + 1
16 Q = np.random.uniform(low=-1, high=1, size=(state_space[0], state_space[1], env.action_space.n))
17
18 def discretize_state(state):
19     state_adj = (state - env.observation_space.low) * np.array([10, 100])
20     return np.round(state_adj, 0).astype(int)
21
22 # Entrenamiento del agente
23 for episode in range(num_episodes):
24     state = discretize_state(env.reset())
25     done = False
26     while not done:
27         if np.random.uniform(0, 1) < epsilon:
28             action = env.action_space.sample() # Exploración
29         else:
30             action = np.argmax(Q[state[0], state[1]]) # Explotación
31
32         next_state, reward, done, _ = env.step(action)
33         next_state = discretize_state(next_state)
34
35 # Actualización de la tabla Q
36 Q[state[0], state[1], action] = Q[state[0], state[1], action] + alpha * (
37     reward + gamma * np.max(Q[next_state[0], next_state[1]]) - Q[state[0], state[1], action])
38
39 state = next_state
40
41 if episode % 1000 == 0:
42     print(f'Episode: {episode}')
43
44 # Prueba del agente entrenado
45 state = discretize_state(env.reset())
46 done = False
47 while not done:
48     action = np.argmax(Q[state[0], state[1]])
49     state, _, done, _ = env.step(action)
50     env.render()
51
52 env.close()
```

Código 7: Código de MountainCar

El cual funciona bien pero no se muestra la parte gráfica a lo que se procedió a hacer otra solicitud.

Solicitud:"Agrega la sección de código en la que se muestre la parte gráfica del entorno"

Código generado:



```
 1 import gym
 2 import numpy as np
 3
 4 # Crear el entorno
 5 env = gym.make('MountainCar-v0')
 6
 7 # Parámetros de aprendizaje
 8 alpha = 0.1 # Tasa de aprendizaje
 9 gamma = 0.99 # Factor de descuento
10 epsilon = 0.1 # Tasa de exploración
11 num_episodes = 10000 # Número de episodios
12
13 # Inicializar la tabla Q
14 state_space = (env.observation_space.high - env.observation_space.low) * np.array([10, 100])
15 state_space = np.round(state_space, 0).astype(int) + 1
16 Q = np.random.uniform(low=-1, high=1, size=(state_space[0], state_space[1], env.action_space.n))
17
18 def discretize_state(state):
19     state_adj = (state - env.observation_space.low) * np.array([10, 100])
20     return np.round(state_adj, 0).astype(int)
21
22 # Entrenamiento del agente
23 for episode in range(num_episodes):
24     state = discretize_state(env.reset())
25     done = False
26     while not done:
27         env.render() # Añadir esta línea para visualizar el entorno
28         if np.random.random() < epsilon:
29             action = env.action_space.sample()
30         else:
31             action = np.argmax(Q[state[0], state[1]])
32
33         next_state, reward, done, _ = env.step(action)
34         next_state = discretize_state(next_state)
35
36         if done and next_state[0] >= env.goal_position:
37             Q[state[0], state[1], action] = reward
38         else:
39             Q[state[0], state[1], action] += alpha * (reward + gamma * np.max(Q[next_state[0], next_state[1]]) - Q[state[0], state[1], action])
40
41         state = next_state
42
43 # Evaluación del agente
44 state = discretize_state(env.reset())
45 done = False
46 while not done:
47     action = np.argmax(Q[state[0], state[1]])
48     state, _, done, _ = env.step(action)
49     env.render()
50
51 env.close()
```

Código 8: Código con parte gráfica MountainCar

Se generaron algunos errores con este código, pero con un **par** de solicitudes más se solucionó y se solicitó agregar la gráfica final, las recompensas en consola y mostrar la parte

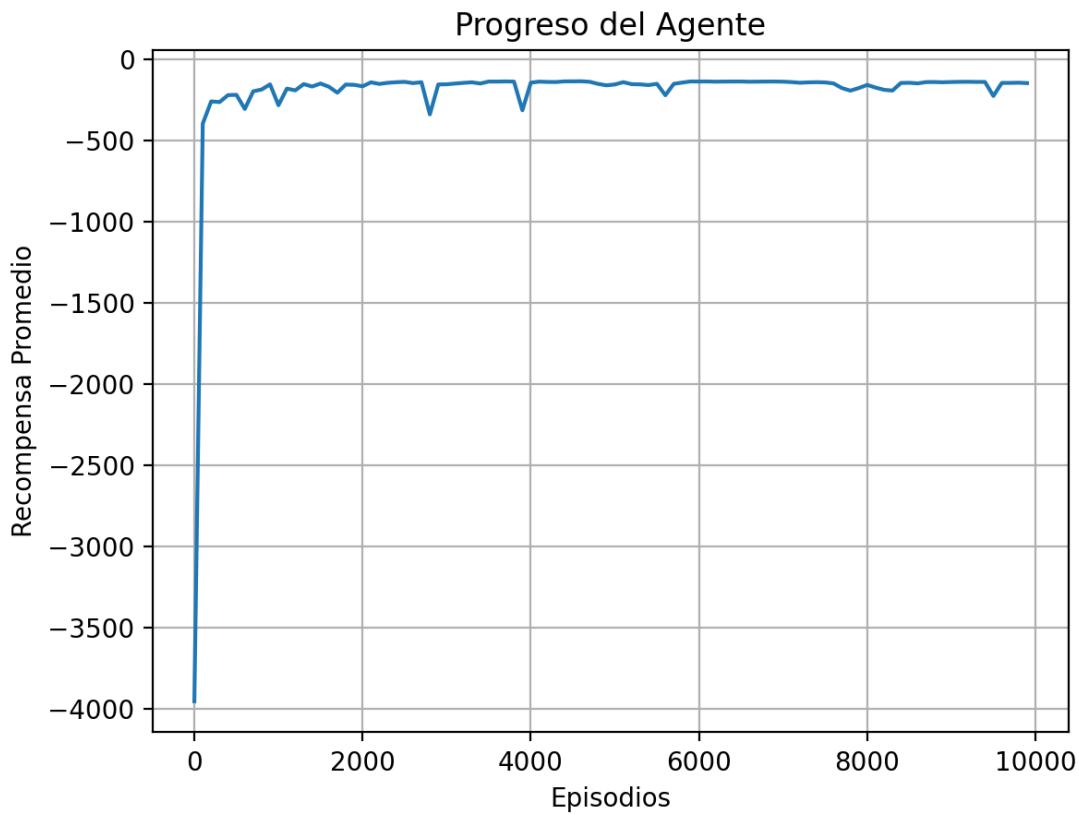
grafica solo al final para agilizar la ejecución, mostrandole como referencia el código del primer agente, quedando de la siguiente manera:

```

● ● ●
1 import gym
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Crear el entorno
6 env = gym.make('MountainCar-v0')
7
8 # Parámetros de aprendizaje
9 alpha = 0.1 # Tasa de aprendizaje
10 gamma = 0.99 # Factor de descuento
11 epsilon = 1.0 # Tasa de exploración inicial
12 epsilon_decay = 0.995
13 epsilon_min = 0.01
14 num_episodes = 10000 # Número de episodios
15
16 # Inicializar la tabla Q
17 state_space = (env.observation_space.high - env.observation_space.low) * np.array([10, 100])
18 state_space = np.round(state_space, 0).astype(int) + 1
19 Q = np.random.uniform(low=-1, high=1, size=(state_space[0], state_space[1], env.action_space.n))
20
21 def discretize_state(state):
22     state_adj = (state - env.observation_space.low) * np.array([10, 100])
23     return np.round(state_adj, 0).astype(int)
24
25 # Listas para monitoreo
26 episode_rewards = []
27
28 # Entrenamiento del agente
29 for episode in range(num_episodes):
30     state = discretize_state(env.reset()[0]) # Extraer la observación del diccionario
31     total_reward = 0
32     done = False
33
34     while not done:
35         if np.random.rand() < epsilon:
36             action = env.action_space.sample()
37         else:
38             action = np.argmax(Q[state[0], state[1]])
39
40         result = env.step(action)
41         next_state = result[0]
42         reward = result[1]
43         done = result[2]
44
45         next_state = discretize_state(next_state)
46
47         if done and next_state[0] >= env.goal_position:
48             Q[state[0], state[1], action] = reward
49         else:
50             Q[state[0], state[1], action] += alpha * (reward + gamma * np.max(Q[next_state[0], next_state[1]]) - Q[state[0], state[1], action])
51
52         state = next_state
53         total_reward += reward
54
55     if done:
56         break
57
58     # Reducir epsilon
59     epsilon = max(epsilon_min, epsilon * epsilon_decay)
60
61     # Guardar las recompensas
62     episode_rewards.append(total_reward)
63
64     if episode % 100 == 0:
65         avg_reward = np.mean(episode_rewards[-100:])
66         print(f"Episodio: {episode}, Recompensa Total: {total_reward}, Recompensa Promedio: {avg_reward}")
67
68     # Evaluación del agente
69     env = gym.make('MountainCar-v0', render_mode='human') # Activar la visualización
70     state = discretize_state(env.reset()[0]) # Extraer la observación del diccionario
71     done = False
72     total_reward = 0
73
74     while not done:
75         action = np.argmax(Q[state[0], state[1]])
76         result = env.step(action)
77         next_state = result[0]
78         reward = result[1]
79         done = result[2]
80         next_state = discretize_state(next_state)
81
82         state = next_state
83         total_reward += reward
84
85     if done:
86         break
87
88     print(f"Recompensa Total en Evaluación: {total_reward}")
89
90     # Cerrar el entorno
91     env.close()
92
93     # Monitoreo y visualización
94     # Calcular recompensas promedio cada 100 episodios
95     episodes = list(range(0, num_episodes, 100))
96     average_rewards = [np.mean(episode_rewards[i:i+100]) for i in range(0, len(episode_rewards), 100)]
97
98     # Graficar el progreso
99     plt.plot(episodes, average_rewards)
100    plt.xlabel('Episodios')
101    plt.ylabel('Recompensa Promedio')
102    plt.title('Progreso del Agente')
103    plt.grid(True)
104    plt.savefig('agent_progress.png')
105    plt.show()
106
107    # Guardar la tabla Q
108    np.save('q_table.npy', Q)

```

Código 9: Código Final MountainCar



Gráfica 3: Grafica de recompensas MountainCar

```
> python /Users/alex/Desktop/Mountaincarv2.py
Episodio: 0, Recompensa Total: -38648.0, Recompensa Promedio: -38648.0
Episodio: 100, Recompensa Total: -414.0, Recompensa Promedio: -3569.19
Episodio: 200, Recompensa Total: -282.0, Recompensa Promedio: -395.45
Episodio: 300, Recompensa Total: -252.0, Recompensa Promedio: -259.97
Episodio: 400, Recompensa Total: -268.0, Recompensa Promedio: -264.43
Episodio: 500, Recompensa Total: -210.0, Recompensa Promedio: -220.94
Episodio: 600, Recompensa Total: -184.0, Recompensa Promedio: -219.7
Episodio: 700, Recompensa Total: -169.0, Recompensa Promedio: -306.72
Episodio: 800, Recompensa Total: -162.0, Recompensa Promedio: -198.01
Episodio: 900, Recompensa Total: -150.0, Recompensa Promedio: -186.95
Episodio: 1000, Recompensa Total: -147.0, Recompensa Promedio: -155.31
Episodio: 1100, Recompensa Total: -157.0, Recompensa Promedio: -283.74
Episodio: 1200, Recompensa Total: -236.0, Recompensa Promedio: -182.38
Episodio: 1300, Recompensa Total: -142.0, Recompensa Promedio: -191.27
Episodio: 1400, Recompensa Total: -137.0, Recompensa Promedio: -153.64
```

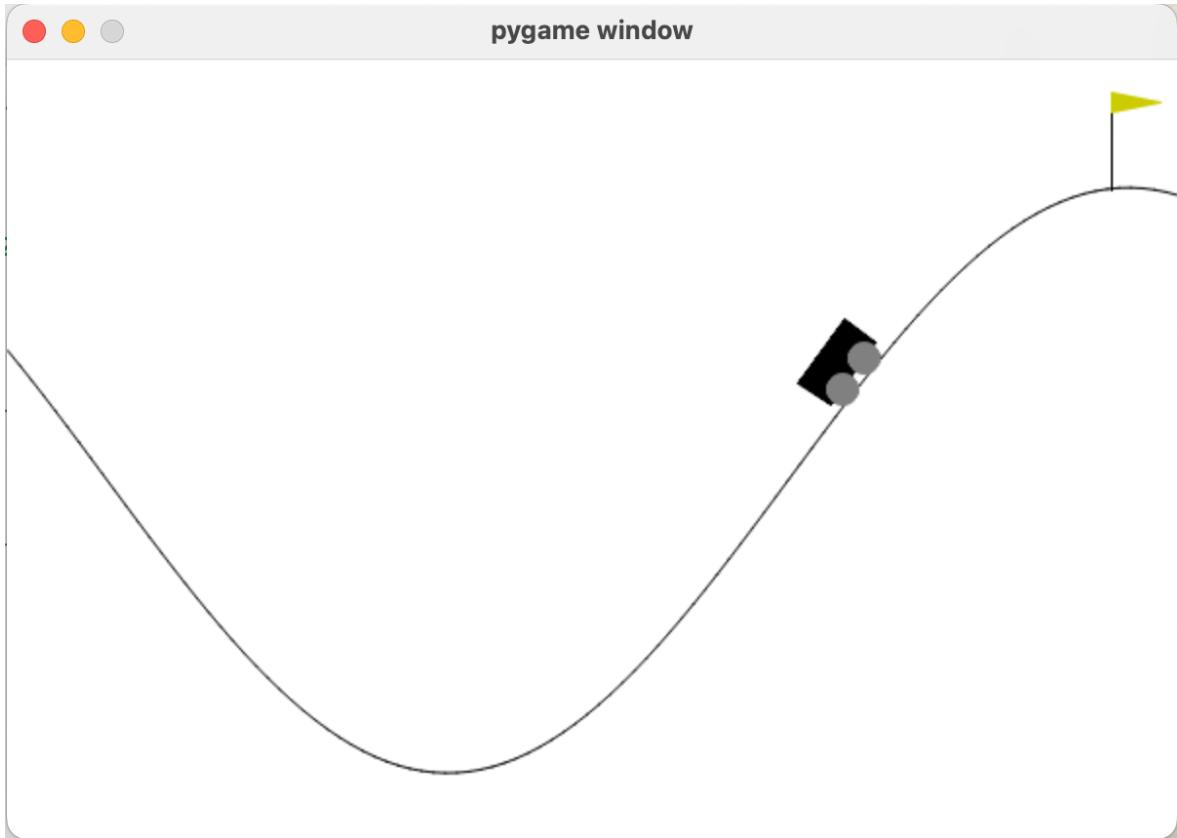
FOTO 3: Recompensas en consola agente MountainCar

```

Episodio: 7900, Recompensa Total: -226.0, Recompensa Promedio: -194.77
Episodio: 8000, Recompensa Total: -227.0, Recompensa Promedio: -178.05
Episodio: 8100, Recompensa Total: -236.0, Recompensa Promedio: -158.85
Episodio: 8200, Recompensa Total: -161.0, Recompensa Promedio: -174.62
Episodio: 8300, Recompensa Total: -232.0, Recompensa Promedio: -189.49
Episodio: 8400, Recompensa Total: -149.0, Recompensa Promedio: -192.77
Episodio: 8500, Recompensa Total: -144.0, Recompensa Promedio: -146.65
Episodio: 8600, Recompensa Total: -139.0, Recompensa Promedio: -145.54
Episodio: 8700, Recompensa Total: -139.0, Recompensa Promedio: -149.31
Episodio: 8800, Recompensa Total: -140.0, Recompensa Promedio: -141.36
Episodio: 8900, Recompensa Total: -143.0, Recompensa Promedio: -141.1
Episodio: 9000, Recompensa Total: -136.0, Recompensa Promedio: -142.51
Episodio: 9100, Recompensa Total: -144.0, Recompensa Promedio: -141.03
Episodio: 9200, Recompensa Total: -144.0, Recompensa Promedio: -139.92
Episodio: 9300, Recompensa Total: -140.0, Recompensa Promedio: -139.5
Episodio: 9400, Recompensa Total: -144.0, Recompensa Promedio: -140.46
Episodio: 9500, Recompensa Total: -142.0, Recompensa Promedio: -140.35
[2] + 20574 killed    python /Users/alex/Desktop/Mountaincarv2.py
Episodio: 9600, Recompensa Total: -150.0, Recompensa Promedio: -226.81
Episodio: 9700, Recompensa Total: -156.0, Recompensa Promedio: -146.61
Episodio: 9800, Recompensa Total: -152.0, Recompensa Promedio: -146.29
Episodio: 9900, Recompensa Total: -138.0, Recompensa Promedio: -145.09
2024-10-08 11:58:12.202 python[23793:12578402] +[IMKClient subclass]: chose IMKClient_Legacy
2024-10-08 11:58:12.202 python[23793:12578402] +[IMKInputSession subclass]: chose IMKInputSession_Legacy
Recompensa Total en Evaluación: -136.0

```

FOTO 4: Recompensas en consola agente MountainCar



Agente 3: Agente MountainCar en ejecución

Tercer agente diseñado con Copilot

Solicitud:"Diseña un agente con entorno Pendulum"

Código generado:



```
1 import gym
2
3 # Crear el entorno
4 env = gym.make('Pendulum-v1')
5
6 # Reiniciar el entorno
7 state = env.reset()
8
9 # Ejecutar una acción en el entorno
10 action = env.action_space.sample() # Tomar una acción aleatoria
11 next_state, reward, done, info = env.step(action)
12
13 # Renderizar el entorno
14 env.render()
15
16 # Cerrar el entorno
17 env.close()
```

Código 10: Primer código del agente pendulum

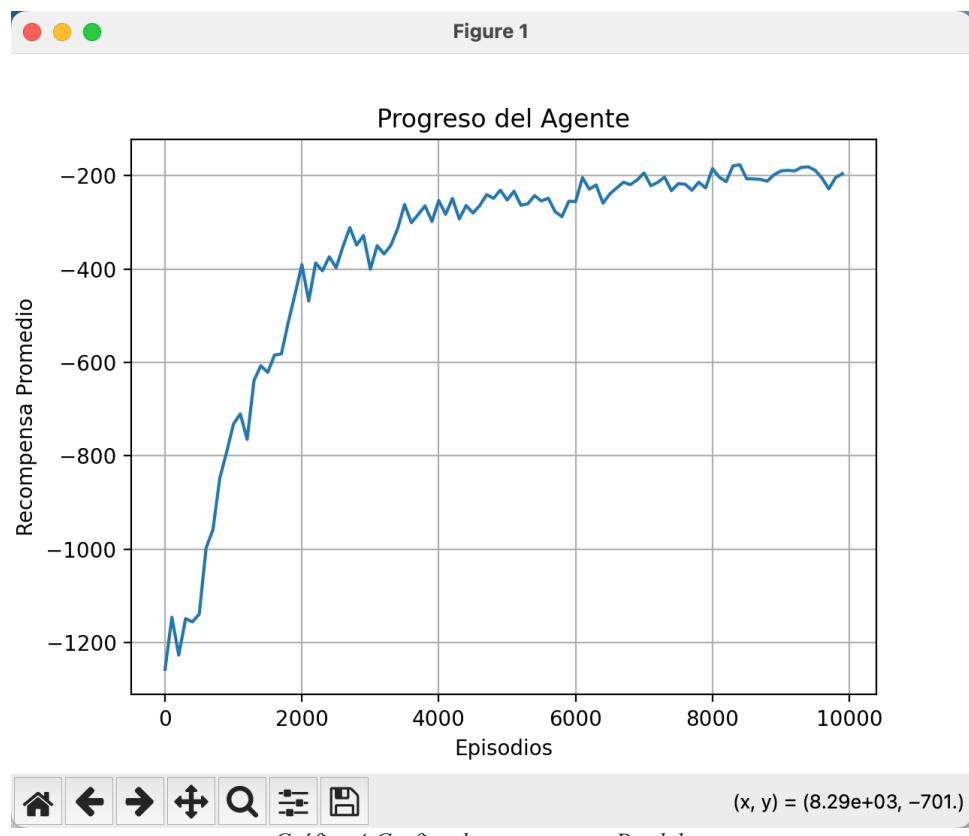
Este código era bastante sencillo y no mostraba la parte gráfica, a lo que se le solicitó basarse en alguno de los agentes anteriores para enriquecer el código con algunas de las funciones de los otros agentes y realizar las modificaciones correspondientes, pero después de 7 solicitudes por errores en el código se obtuvo lo siguiente:

```

 1 import gym
 2 import numpy as np
 3 import matplotlib.pyplot as plt
 4
 5 # Crear el entorno
 6 env = gym.make('Pendulum-v1')
 7
 8 # Parámetros de aprendizaje
 9 alpha = 0.1 # Tasa de aprendizaje
10 gamma = 0.99 # Factor de descuento
11 epsilon = 1.0 # Tasa de exploración inicial
12 epsilon_decay = 0.995
13 epsilon_min = 0.01
14 num_episodes = 100 # Número de episodios
15
16 # Discretización del espacio de estados
17 state_bins = [
18     np.linspace(-1, 1, 20), # theta
19     np.linspace(-1, 1, 20), # theta_dot
20     np.linspace(-8, 8, 20) # torque
21 ]
22
23 def discretize_state(state):
24     state_indices = []
25     for i in range(len(state)):
26         state_indices.append(np.digitize(state[i], state_bins[i]) - 1)
27     return tuple(state_indices)
28
29 # Inicializar la tabla Q
30 state_space_size = tuple(len(bins) for bins in state_bins)
31 action_space_size = 3 # Discretizamos el espacio de acción en 3 acciones: -2, 0, 2
32 Q = np.random.uniform(low=-1, high=1, size=state_space_size + (action_space_size,))
33
34 # Listas para monitoreo
35 episode_rewards = []
36
37 # Entrenamiento del agente
38 for episode in range(num_episodes):
39     state = discretize_state(env.reset()[0]) # Extraer la observación de la tupla
40     total_reward = 0
41     done = False
42
43     while not done:
44         if np.random.rand() < epsilon:
45             action_index = np.random.choice(action_space_size)
46         else:
47             action_index = np.argmax(Q[state])
48
49         action = np.array([action_index * 2 - 2]) # Convertir índice a acción continua (-2, 0, 2)
50         next_state, reward, terminated, truncated, _ = env.step(action)
51         next_state = discretize_state(next_state)
52
53         Q[state][action_index] += alpha * (reward + gamma * np.max(Q[next_state]) - Q[state][action_index])
54
55         state = next_state
56         total_reward += reward
57
58         if terminated or truncated:
59             done = True
60
61     # Reducir epsilon
62     epsilon = max(epsilon_min, epsilon * epsilon_decay)
63
64     # Guardar las recompensas
65     episode_rewards.append(total_reward)
66
67     if episode % 100 == 0:
68         avg_reward = np.mean(episode_rewards[-100:])
69         print(f'Episodio: {episode}, Recompensa Total: {total_reward}, Recompensa Promedio: {avg_reward}')
70
71 # Evaluación del agente
72 env = gym.make('Pendulum-v1', render_mode='human') # Activar la visualización
73 state = discretize_state(env.reset()[0]) # Extraer la observación de la tupla
74 done = False
75 total_reward = 0
76
77 while not done:
78     action_index = np.argmax(Q[state])
79     action = np.array([action_index * 2 - 2]) # Convertir índice a acción continua (-2, 0, 2)
80     next_state, reward, terminated, truncated, _ = env.step(action)
81     next_state = discretize_state(next_state)
82
83     state = next_state
84     total_reward += reward
85
86     if terminated or truncated:
87         done = True
88
89     print(f'Recompensa Total en Evaluación: {total_reward}')
90
91 # Cerrar el entorno
92 env.close()
93
94 # Monitoreo y visualización
95 # Calcular recompensas promedio cada 100 episodios
96 episodes = list(range(0, num_episodes, 100))
97 average_rewards = [np.mean(episode_rewards[i:i + 100]) for i in range(0, len(episode_rewards), 100)]
98
99 # Graficar el progreso
100 plt.plot(episodes, average_rewards)
101 plt.xlabel('Episodios')
102 plt.ylabel('Recompensa Promedio')
103 plt.title('Progreso del Agente')
104 plt.grid(True)
105 plt.savefig('agent_progress.png')
106 plt.show()
107
108 # Guardar la tabla Q
109 np.save('q_table.npy', Q)

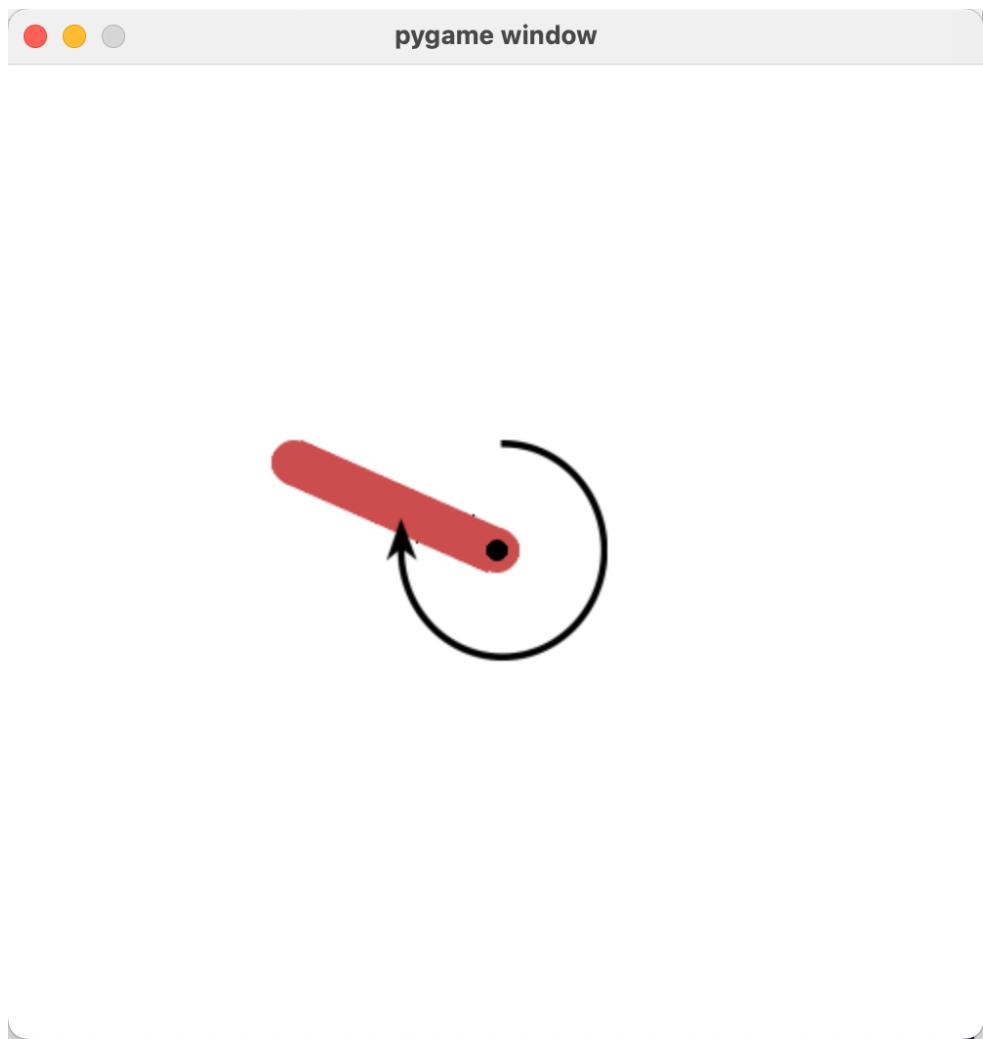
```

Código 11: Código Final de agente Pendulum



```
Episodio: 8000, Recompensa Total: -1.7362471974925122, Recompensa Promedio: -207.32361532952353
Episodio: 8100, Recompensa Total: -126.1649364140624, Recompensa Promedio: -209.70739785979583
Episodio: 8200, Recompensa Total: -123.1646622931862, Recompensa Promedio: -194.54143399893536
Episodio: 8300, Recompensa Total: -123.57079573069372, Recompensa Promedio: -199.33302730295748
Episodio: 8400, Recompensa Total: -237.91145832648542, Recompensa Promedio: -221.00218315495604
Episodio: 8500, Recompensa Total: -124.85865930818552, Recompensa Promedio: -217.28141034731937
Episodio: 8600, Recompensa Total: -122.11683799742337, Recompensa Promedio: -243.56661915260992
Episodio: 8700, Recompensa Total: -2.041520627852826, Recompensa Promedio: -177.3018912965346
Episodio: 8800, Recompensa Total: -523.2907982282026, Recompensa Promedio: -224.8381278087036
Episodio: 8900, Recompensa Total: -1.4535131751696873, Recompensa Promedio: -205.99990578240312
Episodio: 9000, Recompensa Total: -237.73243736481916, Recompensa Promedio: -202.84514882161048
Episodio: 9100, Recompensa Total: -125.44277787504085, Recompensa Promedio: -213.65861201338245
Episodio: 9200, Recompensa Total: -430.626347690928444, Recompensa Promedio: -219.6541158607788
Episodio: 9300, Recompensa Total: -360.23080429683216, Recompensa Promedio: -184.75570081416998
Episodio: 9400, Recompensa Total: -126.96325338734943, Recompensa Promedio: -199.26972353171286
Episodio: 9500, Recompensa Total: -673.0956408544677, Recompensa Promedio: -194.42223101015338
Episodio: 9600, Recompensa Total: -265.5290908944188, Recompensa Promedio: -201.63859024680977
Episodio: 9700, Recompensa Total: -244.0048365632138, Recompensa Promedio: -204.13658501792582
Episodio: 9800, Recompensa Total: -125.17722598001264, Recompensa Promedio: -160.69143494113297
Episodio: 9900, Recompensa Total: -121.2643831248709, Recompensa Promedio: -212.2018408639112
2024-10-08 13:07:05.228 python[87933:12790208] +[IMKClient subclass]: chose IMKClient_Legacy
2024-10-08 13:07:05.228 python[87933:12790208] +[IMKInputSession subclass]: chose IMKInputSession_Legacy
Recompensa Total en Evaluación: -124.17713477692271
[4] 87933 terminated python /Users/alex/Desktop/pendulum.py
```

FOTO 5: Recompensas en pantalla agente Pendulum



Agente 4: Agente Pendulum en ejecución

OpenAI Gym con ROS en Rviz

Agente de prueba en Rviz

Para comenzar a migrar agentes de OpenAI Gym a ROS, es fundamental contar con todas las herramientas y plugins necesarios para ejecutar ROS en el sistema. En mi caso, estoy utilizando un Mac M1, por lo que he seguido el tutorial disponible en el siguiente enlace: [ROS en Mac Silicon \(M1/M2\)](#).

Después de seguir todos los pasos, todo funcionó a la perfección. Sin embargo, me encontré con un inconveniente al intentar ejecutar un script de configuración. Dado que en mi terminal utilizo zsh, traté de ejecutar un archivo .bash, lo que generó algunos errores. Afortunadamente, identifiqué el problema y logré ejecutar todo con normalidad utilizando el archivo setup.zsh, que es el apropiado para mi entorno.

1. Configurar el espacio de trabajo

Primero se crea un espacio de trabajo:

```
mkdir -p ~/catkin_ws/src
```

```
cd ~/catkin_ws/src
```

```
catkin_init_workspace
```

Luego se compila el espacio de trabajo

```
cd ~/catkin_ws
```

```
catkin_make
```

2. Crear un paquete ROS

Crear un nuevo paquete donde se almacenará el código de OpenAI Gym. Con el siguiente comando para crear un paquete que dependa de rospy:

```
cd ~/catkin_ws/src
```

```
catkin_create_pkg gym_agent rospy std_msgs
```

Asegurarse de tener gym o gymnasium en el entorno de ROS

```
pip install gym
```

3. Escribir el script de Gym

Dentro de la carpeta de tu paquete (gym_agent), crear un nuevo directorio llamado scripts:

```
cd ~/catkin_ws/src/gym_agent
```

```
mkdir scripts
```

Luego, crea un archivo Python en el directorio scripts (por ejemplo, gym_agent_node.py)

Hacer que el script sea ejecutable:

```
chmod +x ~/catkin_ws/src/gym_agent/scripts/gym_agent_node.py
```

4. Compilar el paquete

Regresar al directorio raíz del espacio de trabajo Catkin y compilar el paquete:

```
cd ~/catkin_ws
```

```
catkin_make
```

5. Ejecutar el nodo

Asegurarse de que el entorno de ROS esté configurado:

```
source devel/setup.bash o source devel/setup.zsh
```

Dependiendo que terminal se esté utilizando

Luego, ejecutar el nodo:

```
rosrun gym_agent gym_agent_node.py
```

Nombre	Fecha de modificación	Tamaño	Clase
> build	29/10/24, 6:03 p.m.	--	Carpeta
✓ devel	29/10/24, 5:58 p.m.	--	Carpeta
> lib	29/10/24, 5:58 p.m.	--	Carpeta
> share	29/10/24, 5:58 p.m.	--	Carpeta
↳ _setup_util.py	29/10/24, 5:58 p.m.	13 KB	Script Python
↳ cmake.lock	29/10/24, 6:03 p.m.	0 bytes	Documento
↳ env.sh	29/10/24, 5:53 p.m.	506 bytes	Terminal scripts
↳ local_setup.bash	29/10/24, 5:53 p.m.	283 bytes	Script...ell Bash
↳ local_setup.sh	29/10/24, 5:53 p.m.	358 bytes	Terminal scripts
↳ local_setup.zsh	29/10/24, 5:53 p.m.	293 bytes	Terminal scripts
↳ setup.bash	29/10/24, 5:53 p.m.	260 bytes	Script...ell Bash
↳ setup.sh	29/10/24, 5:53 p.m.	3 KB	Terminal scripts
↳ setup.zsh	29/10/24, 5:53 p.m.	270 bytes	Terminal scripts
↳ src	hoy, 1:25 p.m.	--	Carpeta
↳ gym_agent	hoy, 1:25 p.m.	--	Carpeta
↳ scripts	29/10/24, 6:01 p.m.	--	Carpeta
↳ gym_agent_node.py	hoy, 1:03 p.m.	4 KB	Script Python
↳ src	29/10/24, 5:58 p.m.	--	Carpeta
↳ CMakeLists.txt	29/10/24, 6:03 p.m.	7 KB	Texto
↳ package.xml	29/10/24, 5:58 p.m.	3 KB	Texto XML
↳ CMakeLists.txt	29/10/24, 5:53 p.m.	65 bytes	Alias

FOTO 6: Orden de los archivos

El siguiente paso fue traducir el código de los agentes previamente creados al lenguaje de ROS. Para esto, realicé una prueba con el primer agente que había implementado. El código resultante se adaptó para ser compatible con ROS y facilitar la interacción con el entorno.

```

1 #!/usr/bin/env python3
2
3 import rospy
4 from visualization_msgs.msg import Marker, MarkerArray
5 import numpy as np
6 import gymnasium as gym
7
8 class QLearningAgent:
9     def __init__(self, state_space, action_space, learning_rate=0.1, discount_factor=0.99, epsilon=1.0, epsilon_decay=0.995, epsilon_min=0.01):
10         self.state_space = state_space
11         self.action_space = action_space
12         self.learning_rate = learning_rate
13         self.discount_factor = discount_factor
14         self.epsilon = epsilon
15         self.epsilon_decay = epsilon_decay
16         self.epsilon_min = epsilon_min
17         self.q_table = np.zeros(state_space + (action_space.n,))
18
19     def choose_action(self, state):
20         if np.random.rand() < self.epsilon:
21             return self.action_space.sample()
22         else:
23             return np.argmax(self.q_table[state])
24
25     def update_q_table(self, state, action, reward, next_state):
26         best_next_action = np.argmax(self.q_table[next_state])
27         self.q_table[state][action] += self.learning_rate * (reward + self.discount_factor * self.q_table[next_state][best_next_action] - self.q_table[state][action])
28
29     def decay_epsilon(self):
30         self.epsilon = max(self.epsilon_min, self.epsilon * self.epsilon_decay)
31
32     def discretize_state(self):
33         bins = (np.linspace(-2.4, 2.4, num=24),
34                 np.linspace(-2.0, 2.0, num=24),
35                 np.linspace(-0.5, 0.5, num=24),
36                 np.linspace(-0.5, 0.5, num=24))
37
38         state_indices = [np.digitize(state[i], bins[i]) - 1 for i in range(len(state))]
39         return tuple(np.clip(state_indices, 0, 23))
40
41     def publish_cart_position(cart_x):
42         marker_array = MarkerArray()
43         marker = Marker()
44         marker.header.frame_id = "world"
45         marker.type = Marker.CUBE
46         marker.action = Marker.ADD
47         marker.scale.x = 0.5
48         marker.scale.y = 0.1
49         marker.scale.z = 0.1
50         marker.color.a = 1.0
51         marker.color.r = 0.0
52         marker.color.g = 1.0
53         marker.color.b = 0.0
54         marker.pose.position.x = cart_x
55         marker.pose.position.y = 0.0
56         marker.pose.position.z = 0.0
57         marker_array.markers.append(marker)
58         cart_publisher.publish(marker_array)
59
60     def publish_pole_angle(pole_angle, cart_x):
61         marker = Marker()
62         marker.header.frame_id = "world"
63         marker.type = Marker.CYLINDER
64         marker.action = Marker.ADD
65         marker.scale.x = 0.05
66         marker.scale.y = 0.05
67         marker.scale.z = 1.0
68         marker.color.a = 1.0
69         marker.color.r = 1.0
70         marker.color.g = 0.0
71         marker.color.b = 0.0
72         marker.pose.position.x = cart_x
73         marker.pose.position.y = 0.0
74         marker.pose.position.z = 0.5
75         marker.pose.orientation.z = np.sin(pole_angle / 2.0)
76         marker.pose.orientation.w = np.cos(pole_angle / 2.0)
77         pole_publisher.publish(marker)
78
79     if __name__ == "__main__":
80         rospy.init_node('gym_agent_node')
81         cart_publisher = rospy.Publisher('/cartpole/cart_position', MarkerArray, queue_size=10)
82         pole_publisher = rospy.Publisher('/cartpole/pole_angle', Marker, queue_size=10)
83
84         env = gym.make('CartPole-v1')
85         agent = QLearningAgent(state_space=(24, 24, 24, 24), action_space=env.action_space)
86
87         for episode in range(1000):
88             observation = env.reset()[0] # Obtiene directamente la observación inicial sin indexar
89             state = discretize_state(observation)
90             total_reward = 0
91             done = False
92
93             while not done:
94                 action = agent.choose_action(state)
95                 next_observation, reward, done, _ = env.step(action)
96                 next_state = discretize_state(next_observation) # Pasa la observación completa sin indexar
97                 agent.update_q_table(state, action, reward, next_state)
98                 state = next_state
99                 total_reward += reward
100
101             # Extraer posición del carrito y ángulo del poste para visualización
102             cart_x = next_observation[0] # Posición del carrito
103             pole_angle = next_observation[2] # Ángulo del poste
104
105             # Publicar los datos para visualización en RViz
106             publish_cart_position(cart_x)
107             publish_pole_angle(pole_angle, cart_x)
108
109             rospy.sleep(0.05)
110
111             agent.decay_epsilon()
112             rospy.loginfo(f'Episode {episode} - Total Reward: {total_reward}')
113
114         env.close()

```

Código 12: *gym_agent_node.py*

Para ejecutar el agente en el entorno de ROS, se debe abrir una terminal y ejecutar el siguiente comando:

```
rosrun gym_agent gym_agent_node.py
```

En otra terminal, para visualizar gráficamente el agente en el entorno de RViz, se ejecuta:

```
rosrun rviz rviz
```

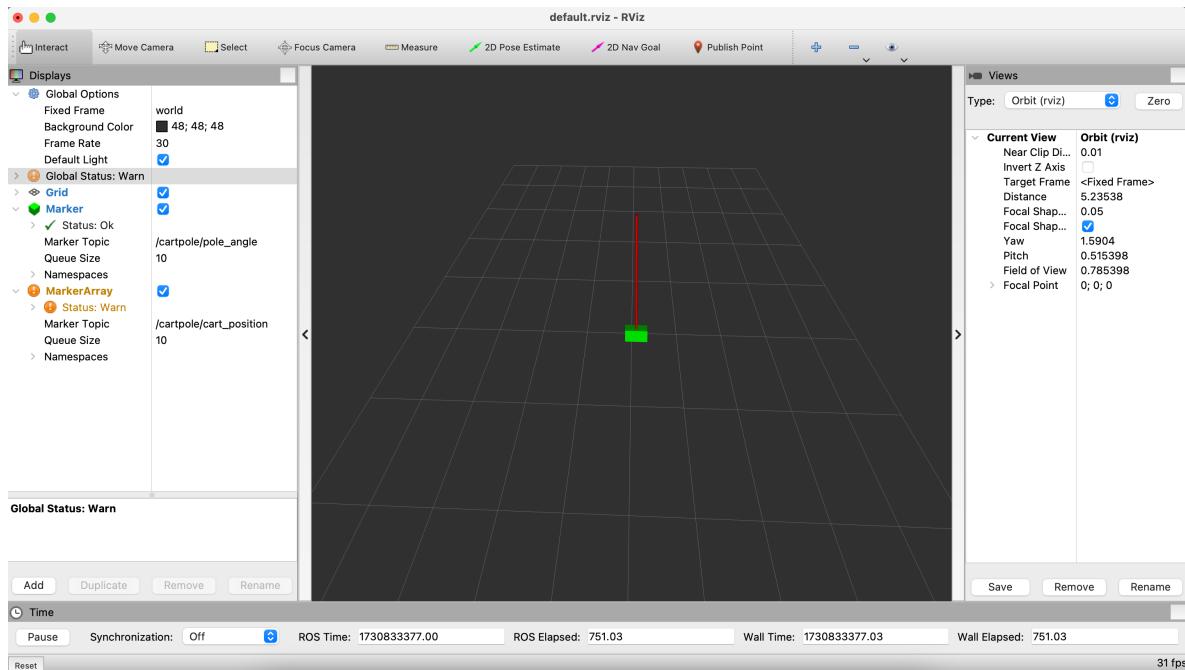


FOTO 7: Cart Pole en Rviz

Una vez que RViz está en funcionamiento, es necesario agregar marcadores en los tópicos correspondientes. En este caso, se utilizan los tópicos /cartpole/pole_angle y /cartpole/cart_position para visualizar la posición del carro y el ángulo del poste, respectivamente.

Este proceso de migración no solo permite ejecutar los agentes en ROS, sino que también proporciona una plataforma robusta para la simulación y visualización en tiempo real, facilitando la comprensión y el análisis del comportamiento del agente en un entorno gráfico.

Agente acrobot en Rviz

Para ejecutar el agente Acrobot en RViz, es necesario modificar el código de manera que este publique la información relevante en los tópicos adecuados. A continuación se presenta el proceso y el código modificado para permitir la visualización de este agente en RViz:

1. Modificar el código del agente:

El primer paso es asegurarse de que el agente esté publicando en los tópicos correspondientes para poder visualizar sus componentes en RViz. El código del agente se debe modificar para que publique las posiciones y movimientos de las partes del Acrobot, como los links y los segmentos del brazo, en los tópicos específicos.

Otorgar los permisos al archivo .py con:

```
chmod +x ~/catkin_ws/src/gym_agent/scripts/acrobot_agent.py
```

```

● ● ●
1 #!/usr/bin/env python3
2
3 import gym
4 import rospy
5 import numpy as np
6 from geometry_msgs.msg import Point
7 from visualization_msgs.msg import Marker
8
9 # Inicializar el nodo de ROS
10 rospy.init_node('acrobot_agent')
11
12 # Crear el entorno de Gym
13 env = gym.make('Acrobot-v1')
14 state = env.reset()
15
16 # Crear publicadores de ROS para los topics de RViz
17 link1_pub = rospy.Publisher('/acrobot/link1', Marker, queue_size=10)
18 link2_pub = rospy.Publisher('/acrobot/link2', Marker, queue_size=10)
19 arm_pub = rospy.Publisher('/acrobot/arm', Marker, queue_size=10)
20 rate = rospy.Rate(30) # Frecuencia de publicación en Hz
21
22 # Función para crear un marcador de RViz
23 def create_marker(position, marker_id, color):
24     marker = Marker()
25     marker.header.frame_id = "world"
26     marker.type = Marker.SPHERE
27     marker.action = Marker.ADD
28     marker.pose.position = Point(*position)
29     marker.pose.orientation.w = 1.0
30     marker.scale.x = 0.1
31     marker.scale.y = 0.1
32     marker.scale.z = 0.1
33     marker.color.r, marker.color.g, marker.color.b, marker.color.a = color
34     marker.id = marker_id
35     return marker
36
37 # Función para crear un marcador de línea para el brazo
38 def create_arm_marker(start_pos, end_pos):
39     marker = Marker()
40     marker.header.frame_id = "world"
41     marker.type = Marker.LINE_STRIP
42     marker.action = Marker.ADD
43     marker.scale.x = 0.05 # Grosor de la linea
44     marker.color.r = 0.0
45     marker.color.g = 1.0
46     marker.color.b = 0.0
47     marker.color.a = 1.0
48     marker.points = [Point(*start_pos), Point(*end_pos)]
49     marker.id = 3
50     return marker
51
52 # Bucle principal del agente (se ejecutará indefinidamente hasta que lo detengas manualmente)
53 while not rospy.is_shutdown():
54     # Seleccionar una acción aleatoria
55     action = env.action_space.sample()
56
57     # Ejecutar la acción en el entorno
58     next_state, reward, terminated, truncated, info = env.step(action)
59
60     # Calcular las posiciones de los enlaces del Acrobot
61     # Longitud de las barras del Acrobot
62     l1, l2 = 1.0, 1.0
63     link1_pos = [l1 * np.cos(next_state[0]), l1 * np.sin(next_state[0]), 0.0]
64     link2_pos = [
65         link1_pos[0] + l2 * np.cos(next_state[0] + next_state[1]),
66         link1_pos[1] + l2 * np.sin(next_state[0] + next_state[1]),
67         0.0
68     ]
69
70     # Crear y publicar los marcadores en RViz
71     marker1 = create_marker(link1_pos, marker_id=1, color=(1.0, 0.0, 0.0, 1.0))
72     marker2 = create_marker(link2_pos, marker_id=2, color=(0.0, 0.0, 1.0, 1.0))
73     arm_marker = create_arm_marker([0.0, 0.0, 0.0], link1_pos)
74     arm_marker.points.append(Point(*link2_pos))
75     link1_pub.publish(marker1)
76     link2_pub.publish(marker2)
77     arm_pub.publish(arm_marker)
78
79     # Esperar según la frecuencia establecida
80     rate.sleep()
81
82     # Cerrar el entorno cuando se detiene el nodo
83     env.close()

```

Código 13: *Acrobot_agent.py*

2. Ejecutar el agente en ROS:

Una vez que el código esté listo, se debe abrir una terminal y ejecutar el siguiente comando para lanzar el agente en el entorno de ROS:

```
rosrun gym_agent acrobot_agent.py
```

3. Visualizar el agente en RViz:

Para visualizar gráficamente el comportamiento del agente en RViz, se debe abrir una segunda terminal y ejecutar el siguiente comando:

```
rosrun rviz rviz
```

4. Configuración en RViz:

Una vez que RViz esté abierto, es necesario agregar los tópicos correspondientes que permiten visualizar las distintas partes del Acrobot. Para esto, se deben agregar los siguientes tópicos en RViz:

- /acrobot/link1: Representa el primer segmento del brazo del Acrobot.
- /acrobot/link2: Representa el segundo segmento del brazo del Acrobot.
- /acrobot/arm: Representa el conjunto del brazo del Acrobot.

Estos tópicos permiten visualizar las posiciones y movimientos de las partes del Acrobot en el entorno de simulación de RViz, proporcionando una representación gráfica del comportamiento del agente en tiempo real.

Este proceso no solo facilita la visualización del agente, sino que también ayuda a comprender mejor cómo interactúa el agente con el entorno y cómo sus movimientos afectan el desempeño de la tarea que está aprendiendo.

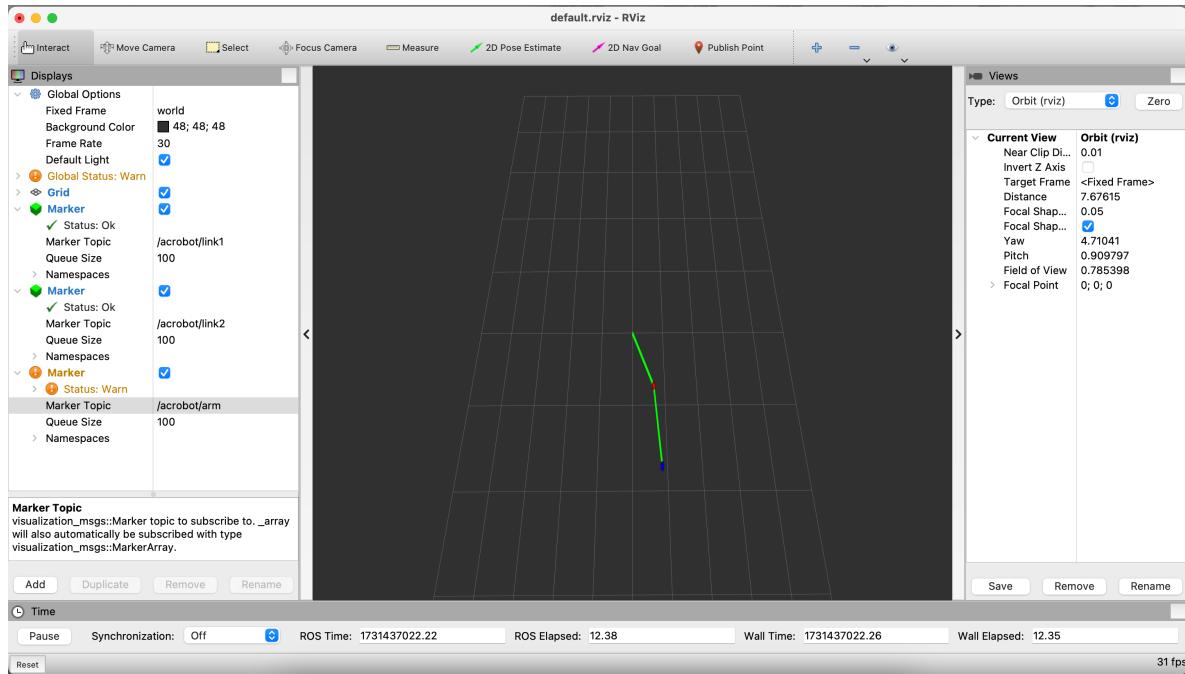


FOTO 8: Acrobot en Rviz

Agente Mountain Car en Rviz

Para ejecutar el agente Mountain Car en RViz, es necesario modificar el código de manera que este publique la información relevante en los tópicos adecuados. A continuación se presenta el proceso y el código modificado para permitir la visualización de este agente en RViz:

1. Modificar el código del agente:

El primer paso es asegurarse de que el agente esté publicando en los tópicos correspondientes para poder visualizar sus componentes en RViz. El código del agente se debe modificar para que publique las posiciones y movimientos de las partes del agente Mountain Car, como la carretera y el carro, en los tópicos específicos.

Otorgar los permisos al archivo .py con:

```
chmod +x ~/catkin_ws/src/gym_agent/scripts/mountaincar_agent.py
```

```

● ○ ●
1 #!/usr/bin/env python3
2
3 import rospy
4 from visualization_msgs.msg import Marker
5 from geometry_msgs.msg import Point
6 import gymnasium as gym
7 import numpy as np
8
9 # Inicializar el nodo de ROS
10 rospy.init_node('mountaincar_visualizer')
11
12 # Crear los publicadores
13 car_publisher = rospy.Publisher('/mountaincar/car_position', Marker, queue_size=10)
14 track_publisher = rospy.Publisher('/mountaincar/track', Marker, queue_size=10)
15
16 # Crear el entorno de MountainCar
17 env = gym.make('MountainCar-v0')
18 env = env.unwrapped # Desempaquetar el entorno base para acceder a los límites
19 min_position = env.min_position
20 max_position = env.max_position
21
22 # Configurar y publicar el marcador del carro
23 def publish_car(position):
24     car_marker = Marker()
25     car_marker.header.frame_id = "world"
26     car_marker.header.stamp = rospy.Time.now()
27     car_marker.ns = "mountaincar"
28     car_marker.id = 0
29     car_marker.type = Marker.CUBE
30     car_marker.action = Marker.ADD
31     car_marker.scale.x = 0.1
32     car_marker.scale.y = 0.05
33     car_marker.scale.z = 0.05
34     car_marker.color.a = 1.0
35     car_marker.color.r = 1.0
36     car_marker.color.g = 0.0
37     car_marker.color.b = 0.0
38     car_marker.pose.position.x = position
39     car_marker.pose.position.y = 0.0
40     car_marker.pose.position.z = np.sin(3 * position) # Alinear el carro con la carretera
41     car_marker.pose.orientation.x = 0.0
42     car_marker.pose.orientation.y = 0.0
43     car_marker.pose.orientation.z = 0.0
44     car_marker.pose.orientation.w = 1.0
45     car_publisher.publish(car_marker)
46
47 # Configurar y publicar el marcador de la carretera
48 def publish_track():
49     track_marker = Marker()
50     track_marker.header.frame_id = "world"
51     track_marker.header.stamp = rospy.Time.now()
52     track_marker.ns = "mountaincar"
53     track_marker.id = 1
54     track_marker.type = Marker.LINE_STRIP
55     track_marker.action = Marker.ADD
56     track_marker.scale.x = 0.02
57     track_marker.color.a = 1.0
58     track_marker.color.r = 0.0
59     track_marker.color.g = 1.0
60     track_marker.color.b = 0.0
61
62     for x in np.linspace(min_position, max_position, 100):
63         y = np.sin(3 * x)
64         point = Point()
65         point.x = x
66         point.y = 0.0
67         point.z = y # Establecer la carretera en el eje Z
68         track_marker.points.append(point)
69
70     track_publisher.publish(track_marker)
71
72 # Simulación principal
73 rate = rospy.Rate(10) # 10 Hz
74 state, _ = env.reset() # Reiniciar el entorno
75 done = False
76
77 while not rospy.is_shutdown() and not done:
78     # Acción aleatoria solo para visualización; puedes cambiarla por tu agente
79     action = env.action_space.sample()
80     next_state, reward, done, _, _ = env.step(action)
81
82     # Publicar la posición del carro alineada con la carretera
83     publish_car(next_state[0])
84
85     # Publicar la carretera (una sola vez es suficiente)
86     publish_track()
87
88     rate.sleep()

```

Código 14: Mountaincar_agent.py

2. Ejecutar el agente en ROS:

Una vez que el código esté listo, se debe abrir una terminal y ejecutar el siguiente comando para lanzar el agente en el entorno de ROS:

```
rosrun gym_agent mountaincar_agent.py
```

3. Visualizar el agente en RViz:

Para visualizar gráficamente el comportamiento del agente en RViz, se debe abrir una segunda terminal y ejecutar el siguiente comando:

```
rosrun rviz rviz
```

4. Configuración en RViz:

Una vez que RViz esté abierto, es necesario agregar los tópicos correspondientes que permiten visualizar las distintas partes del Acrobot. Para esto, se deben agregar los siguientes tópicos en RViz:

- /mountaincar/car_position: Representa la posición del coche con un cubo.
- /mountaincar/track: Representa la carretera.

Estos tópicos permiten visualizar la posición y movimiento de las partes del agente Mountain Car en el entorno de simulación de RViz, proporcionando una representación gráfica del comportamiento del agente en tiempo real.

Este proceso no solo facilita la visualización del agente, sino que también ayuda a comprender mejor cómo interactúa el agente con el entorno y cómo sus movimientos afectan el desempeño de la tarea que está aprendiendo.

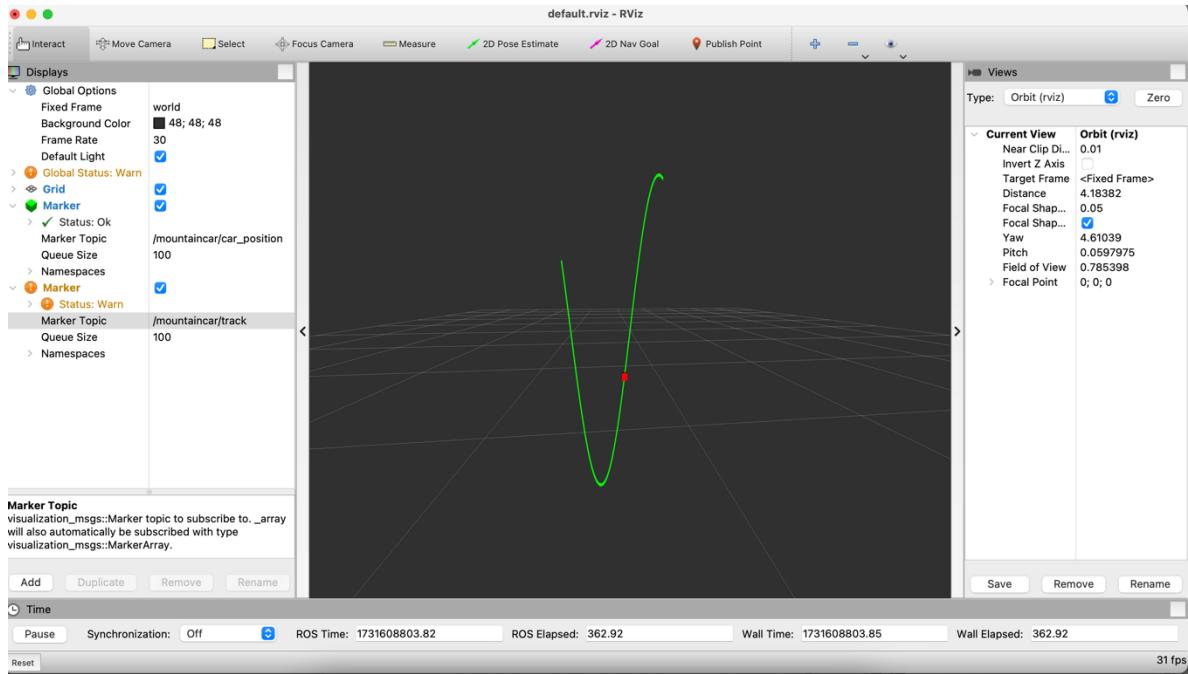


FOTO 9: Mountain Car en Rviz

Agente pendulum en Rviz

Para ejecutar el agente Pendulum en RViz, es necesario modificar el código de manera que este publique la información relevante en los tópicos adecuados. A continuación se presenta el proceso y el código modificado para permitir la visualización de este agente en RViz:

1. Modificar el código del agente:

El primer paso es asegurarse de que el agente esté publicando en los tópicos correspondientes para poder visualizar sus componentes en RViz. El código del agente se debe modificar para que publique las posiciones y movimientos de las partes del agente Pendulum, en los tópicos específicos.

Otorgar los permisos al archivo .py con:

```
chmod +x ~/catkin_ws/src/gym_agent/scripts/pendulum_agent.py
```

```

1 #!/usr/bin/env python3
2
3 import rospy
4 from visualization_msgs.msg import Marker
5 from geometry_msgs.msg import Point
6 import gymnasium as gym
7 import numpy as np
8
9 # Inicializar el nodo de ROS
10 rospy.init_node('pendulum_visualizer')
11
12 # Crear publicadores
13 pendulum_publisher = rospy.Publisher('/pendulum/pendulum_marker', Marker, queue_size=1)
14 tip_publisher = rospy.Publisher('/pendulum/pendulum_tip', Marker, queue_size=10)
15
16 # Crear el entorno de Pendulum
17 env = gym.make('Pendulum-v1')
18 env = env.unwrapped # Desempaquetar el entorno para un control más directo
19
20 # Configurar y publicar el marcador del péndulo
21 def publish_pendulum(theta):
22     # Publicar el marcador de la línea del péndulo
23     pendulum_marker = Marker()
24     pendulum_marker.header.frame_id = "world"
25     pendulum_marker.header.stamp = rospy.Time.now()
26     pendulum_marker.ns = "pendulum"
27     pendulum_marker.id = 0
28     pendulum_marker.type = Marker.LINE_STRIP
29     pendulum_marker.action = Marker.ADD
30     pendulum_marker.scale.x = 0.05 # Grosor de la linea
31     pendulum_marker.color.a = 1.0
32     pendulum_marker.color.r = 0.0
33     pendulum_marker.color.g = 1.0
34     pendulum_marker.color.b = 0.0
35
36     # Asegurarse de que la gravedad está dirigida hacia abajo en el eje z
37     pendulum_length = 1.0 # Longitud del péndulo
38     x = pendulum_length * np.sin(theta) # Movimiento en el plano XY
39     z = -pendulum_length * np.cos(theta)
40
41     # Crear puntos para la línea del péndulo
42     base = Point()
43     base.x = 0.0
44     base.y = 0.0
45     base.z = 0.0
46
47     end = Point()
48     end.x = x
49     end.y = 0.0 # No hay movimiento en el eje Y, solo X y Z
50     end.z = z
51
52     pendulum_marker.points.append(base)
53     pendulum_marker.points.append(end)
54
55     # Publicar el marcador de la linea
56     pendulum_publisher.publish(pendulum_marker)
57
58     # Publicar el marcador del círculo en la punta del péndulo
59     tip_marker = Marker()
60     tip_marker.header.frame_id = "world"
61     tip_marker.header.stamp = rospy.Time.now()
62     tip_marker.ns = "pendulum_tip"
63     tip_marker.id = 1
64     tip_marker.type = Marker.SPHERE
65     tip_marker.action = Marker.ADD
66     tip_marker.scale.x = 0.1 # Radio del círculo
67     tip_marker.scale.y = 0.1
68     tip_marker.scale.z = 0.1
69     tip_marker.color.a = 1.0
70     tip_marker.color.r = 1.0
71     tip_marker.color.g = 0.0
72     tip_marker.color.b = 0.0
73
74     # Posición del círculo (punta del péndulo)
75     tip_marker.pose.position.x = x
76     tip_marker.pose.position.y = 0.0
77     tip_marker.pose.position.z = z
78
79     # Publicar el marcador del círculo
80     tip_publisher.publish(tip_marker)
81
82     # Simulación principal
83     rate = rospy.Rate(10) # 10 Hz
84     state, _ = env.reset() # Reiniciar el entorno
85     done = False
86
87     while not rospy.is_shutdown() and not done:
88         # Acción aleatoria solo para visualización; puedes cambiarla por tu agente
89         action = env.action_space.sample()
90         next_state, reward, done, _, _ = env.step(action)
91
92         # Publicar la posición del péndulo
93         theta = next_state[0] # Ángulo del péndulo
94         publish_pendulum(theta)
95
96         rate.sleep()

```

Código 15: Pendulum_Agent.py

1. Ejecutar el agente en ROS:

Una vez que el código esté listo, se debe abrir una terminal y ejecutar el siguiente comando para lanzar el agente en el entorno de ROS:

```
rosrun gym_agent pendulum_agent.py
```

2. Visualizar el agente en RViz:

Para visualizar gráficamente el comportamiento del agente en RViz, se debe abrir una segunda terminal y ejecutar el siguiente comando:

```
rosrun rviz rviz
```

3. Configuración en RViz:

Una vez que RViz esté abierto, es necesario agregar los tópicos correspondientes que permiten visualizar las distintas partes del Pendulum. Para esto, se deben agregar los siguientes tópicos en RViz:

- /pendulum/pendulum_marker: Representa la linea del pendulo.
- /pendulum/pendulum_tip: Representa la punta.

Estos tópicos permiten visualizar la posición y movimiento de las partes del agente Pendulum en el entorno de simulación de RViz, proporcionando una representación gráfica del comportamiento del agente en tiempo real.

Este proceso no solo facilita la visualización del agente, sino que también ayuda a comprender mejor cómo interactúa el agente con el entorno y cómo sus movimientos afectan el desempeño de la tarea que está aprendiendo.

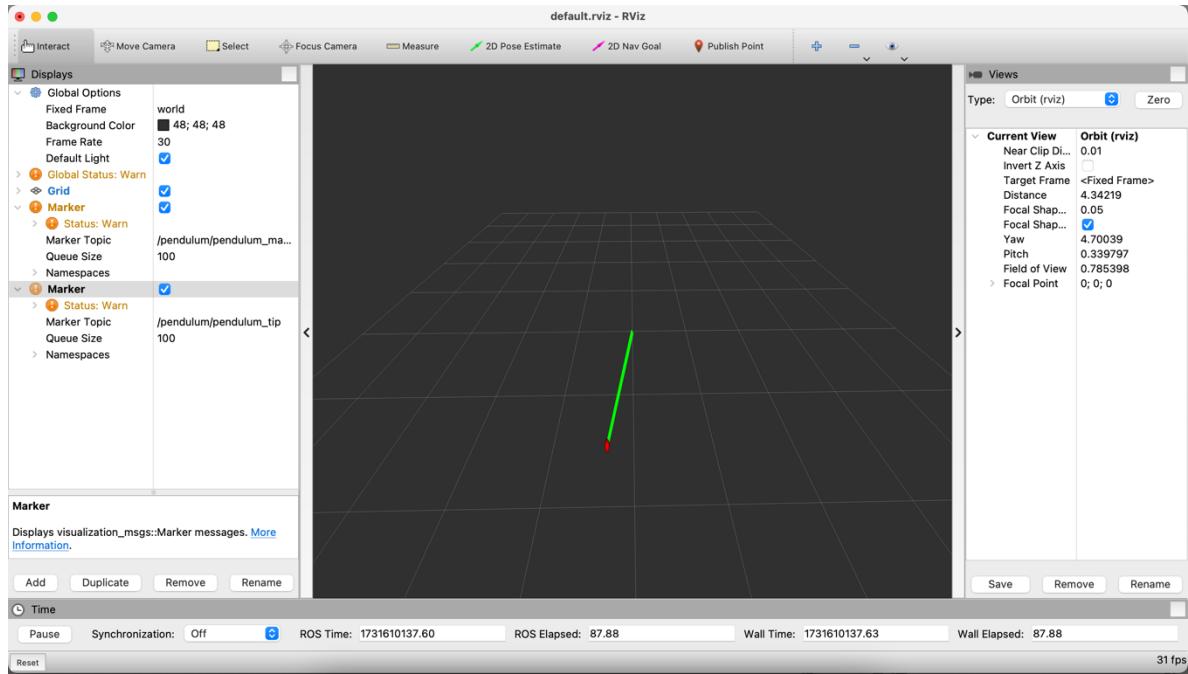


FOTO 10:Pendulum en Rviz

Github en donde se encuentran los códigos de los agentes para ser ejecutados en Rviz:

<https://github.com/RousingManxx/AgentesOpenAIGymRviz>

Agentes entrenados para visualización en RViz

A continuación se presentan los agentes desarrollados, sus características principales y los resultados obtenidos al integrarlos en **RViz** tras aplicar el entrenamiento utilizando algoritmos básicos como **Q-Learning**.

Agente Acrobot con entrenamiento

Ejecución y resultados:

- En consola, se muestran estadísticas del entrenamiento, incluyendo recompensas por episodio.
- En RViz, se observa cómo el agente aprende a balancear el brazo.
- Consola:

```
Episodio: 100, Recompensa Promedio: -497.13
Episodio: 200, Recompensa Promedio: -461.13
Episodio: 300, Recompensa Promedio: -435.46
Episodio: 400, Recompensa Promedio: -409.80
Episodio: 500, Recompensa Promedio: -367.82
Episodio: 600, Recompensa Promedio: -372.64
Episodio: 700, Recompensa Promedio: -321.98
Episodio: 800, Recompensa Promedio: -294.38
Episodio: 900, Recompensa Promedio: -316.82
Episodio: 1000, Recompensa Promedio: -302.09
Entrenamiento completado. Visualizando agente entrenado en RViz...
```

FOTO 11: Recompensas en consola Acrobot_trained

- Código:

```

1  #!/usr/bin/env python3
2
3  import time
4  import math
5  import rospy
6  import numpy as np
7  from geometry_msgs.msg import Point
8  from visualization_msgs.msg import Marker
9
10
11  # Parámetros de Q-learning
12  state_space = (6, 6, 6, 6, 6, 6) # Ajustar la discretización del espacio de estados
13  action_space = 3
14  learning_rate = 0.1
15  discount_factor = 0.99
16  epsilon = 1.0
17  epsilon_min = 0.01
18  num_episodes = 1000
19
20
21  # Inicialización de la tabla Q
22  q_table = np.zeros(state_space + (action_space,))
23
24  # Función para discretizar el estado
25  def discretize_state(state):
26      bins = [
27          np.linspace(-1.0, 1.0, numstate_space[0]), # cos(theta1)
28          np.linspace(-1.0, 1.0, numstate_space[1]), # sin(theta1)
29          np.linspace(-1.0, 1.0, numstate_space[2]), # cos(theta2)
30          np.linspace(-1.0, 1.0, numstate_space[3]), # sin(theta2)
31          np.linspace(-4.0, 4.0, numstate_space[4]), # theta1_dot
32          np.linspace(-9.0, 9.0, numstate_space[5]) # theta2_dot
33      ]
34      indices = [np.digitize(state[i], bins[i]) - 1 for i in range(len(state))]
35      return tuple(np.array(indices, dtype=np.uint8))
36
37  # Función para crear marcadores de RViz
38  def create_ar_marker(position, marker_id, color):
39      marker = Marker()
40      marker.header.frame_id = "world"
41      marker.type = Marker.SPHERE
42      marker.action = Marker.ADD
43      marker.pose.position = Point(position)
44      marker.pose.orientation.w = 1.0
45      marker.scale.x = 0.1
46      marker.scale.y = 0.1
47      marker.scale.z = 0.1
48      marker.id = marker_id
49      return marker
50
51  def create_ar_stripe(start_pos, end_pos):
52      marker = Marker()
53      marker.header.frame_id = "world"
54      marker.type = Marker.LINE_STRIP
55      marker.action = Marker.ADD
56      marker.scale.x = 0.05
57      marker.scale.y = 0.05
58      marker.color.g = 1.0
59      marker.color.b = 0.0
60      marker.color.a = 1.0
61      marker.points = [Point(start_pos), Point(end_pos)]
62      return marker
63
64  # Entrenamiento del agente
65  env = gym.make('Acrobot-v1')
66  episode_rewards = []
67
68  print("Entrenando agente...")
69
70  for episode in range(num_episodes):
71      obs = env.reset() # Actualizado para obtener el estado inicial
72      state = discretize_state(obs)
73      total_reward = 0
74      done = False
75
76      while not done:
77          # Seleccionar una acción (exploración/explotación)
78          if np.random.rand() < epsilon:
79              action = env.action_space.sample()
80          else:
81              action = np.argmax(q_table[state])
82
83          # Realizar la acción
84          obs, reward, terminated, truncated, info = env.step(action) # Actualizado
85          next_state = discretize_state(obs)
86          best_next_action = np.argmax(q_table[next_state])
87
88          # Actualizar la tabla Q
89          q_table[state[action]] += learning_rate * (
90              reward + discount_factor * q_table[next_state][best_next_action] - q_table[state[action]]
91          )
92          state = next_state
93          total_reward += reward
94
95          if terminated or truncated:
96              done = True
97
98          # Reducir epsilon
99          epsilon = max(epsilon_min, epsilon * epsilon_decay)
100
101      # Guardar la recompensa total
102      episode_rewards.append(total_reward)
103
104      # Mostrar el progreso cada 100 episodios
105      if (episode + 1) % 100 == 0:
106          avg_reward = np.mean(episode_rewards[-100:])
107          print(f"Episodio: {episode + 1}, Recompensa Promedio: {avg_reward:.2f}")
108
109      # Guardar la tabla Q entrenada
110      np.save('q_table_acrobot.npy', q_table)
111      print("Entrenamiento completado. Visualizando agente entrenado en RViz...")
112
113  # Publicadores de RViz
114  # Link1_pub = rospy.Publisher('/acrobot/link1', Marker, queue_size=10)
115  link1_pub = rospy.Publisher('/acrobot/link1', Marker, queue_size=10)
116  link2_pub = rospy.Publisher('/acrobot/link2', Marker, queue_size=10)
117  arm_pub = rospy.Publisher('/acrobot/arm', Marker, queue_size=10)
118  rate = rospy.Rate(30)
119
120  # Evaluación y visualización del agente entrenado
121  obs_ = env.reset() # Actualizado
122  state = discretize_state(obs_)
123  done = False
124
125  while not rospy.is_shutdown() and not done:
126      action = np.argmax(q_table[state])
127      next_state, reward, terminated, _ = env.step(action)
128      next_state = discretize_state(next_state)
129
130      # Posiciones del Acrobot
131      l1, l2 = 1.0, 1.0
132      link1_pos = [l1 * np.sin(env.state[0]), -l1 * np.cos(env.state[0]), 0.0]
133      link2_pos = [link1_pos[0] + l2 * np.sin(env.state[0] + env.state[1]),
134                  link1_pos[1] + l2 * np.cos(env.state[0] + env.state[1]),
135                  0.0]
136
137      marker1 = create_marker(link1_pos, marker_id=1, color=(1.0, 0.0, 0.0, 1.0))
138      marker2 = create_marker(link2_pos, marker_id=2, color=(0.0, 0.0, 1.0, 1.0))
139      arm_marker = create_ar_marker([0.0, 0.0, 0.0], link1_pos)
140      arm_marker.points.append(Point(link2_pos))
141      link1_pub.publish(marker1)
142      link2_pub.publish(marker2)
143      arm_pub.publish(arm_marker)
144
145      state = next_state
146      done = terminated or truncated
147      rate.sleep()
148
149  env.close()

```

Código 16: Código Acrobot_trained

Agente Mountain Car con entrenamiento

Ejecución y resultados:

- Se visualiza el coche desplazándose en RViz.
- En consola, se registran métricas como la recompensa acumulada y el número de pasos por episodio.

- Consola:

```
> rosrun gym_agent mountaincar_trained.py
Episodio 100, Recompensa promedio: -200.00
Episodio 200, Recompensa promedio: -200.00
Episodio 300, Recompensa promedio: -200.00
Episodio 400, Recompensa promedio: -200.00
Episodio 500, Recompensa promedio: -200.00
Episodio 600, Recompensa promedio: -199.07
Episodio 700, Recompensa promedio: -199.23
Episodio 800, Recompensa promedio: -198.85
Episodio 900, Recompensa promedio: -196.23
Episodio 1000, Recompensa promedio: -190.57
Entrenamiento completado.
```

FOTO 12: Recompensas en consola Mountaincar_trained

- Código:

```

1  #!/usr/bin/env python3
2
3  import rospy
4  import gymnasium as gym
5  import numpy as np
6  from geometry_msgs.msg import Point
7  from visualization_msgs.msg import Marker
8
9
10 # Configuración de parámetros de Q-learning
11 state_space = (20, 20) # Más granularidad para la discretización del espacio de estados
12 action_space = 3 # Tres acciones disponibles en MountainCar
13 learning_rate = 0.1
14 discount_factor = 0.99
15 epsilon = 1.0
16 epsilon_decay = 0.999
17 epsilon_min = 0.01
18 num_episodes = 1000
19
20 # Inicialización de la tabla Q
21 q_table = np.zeros(state_space + (action_space,))
22
23 # Discretización del espacio de estados
24 def discretize_state(state):
25     bins = [
26         np.linspace(-1.2, 0.6, state_space[0]), # Posición
27         np.linspace(-0.07, 0.07, state_space[1]) # Velocidad
28     ]
29     indices = [np.digitize(state[i], bins[i]) - 1 for i in range(len(state))]
30     return tuple(np.array(indices, dtype=np.int32))
31
32 # Crear marcador para el carro
33 def create_car_marker(position):
34     marker = Marker()
35     marker.header.frame_id = "world"
36     marker.header.stamp = rospy.Time.now()
37     marker.ns = "mountaincar"
38     marker.id = 0
39     marker.type = Marker.CUBE
40     marker.action = Marker.ADD
41     marker.pose.position.x = position[0]
42     marker.pose.position.y = 0.0
43     marker.pose.position.z = np.sin(3 * position[1])
44     marker.scale.x = 0.1
45     marker.scale.y = 0.1
46     marker.scale.z = 0.1
47     marker.color.r = 1.0
48     marker.color.g = 0.0
49     marker.color.b = 0.0
50     marker.color.a = 1.0
51     return marker
52
53 # Crear marcador para la carretera
54 def create_track_marker():
55     marker = Marker()
56     marker.header.frame_id = "world"
57     marker.header.stamp = rospy.Time.now()
58     marker.ns = "mountaincar"
59     marker.type = Marker.LINE_STRIP
60     marker.action = Marker.ADD
61     marker.scale.x = 0.05
62     marker.color.r = 0.0
63     marker.color.g = 0.0
64     marker.color.b = 0.0
65     marker.color.a = 1.0
66
67     for x in np.linspace(-1.2, 0.6, 100):
68         point = Point()
69         point.x = x
70         point.y = 0.0
71         point.z = 0.0
72         marker.points.append(point)
73
74     return marker
75
76 # Entrenamiento del agente
77 env = gym.make('MountainCar-v0')
78 rewards = []
79
80 for episode in range(num_episodes):
81     state, _ = env.reset()
82     discretized_state = discretize_state(state)
83     total_reward = 0
84     done = False
85
86     while not done:
87         # Selección de acción (exploración/explotación)
88         if np.random.rand() < epsilon:
89             action = env.action_space.sample()
90         else:
91             action = np.argmax(q_table[discretized_state])
92
93         # Ejecución de la acción
94         next_state, reward, terminated, truncated, _ = env.step(action)
95         next_discretized_state = discretize_state(next_state)
96
97         # Actualización de Q-table
98         best_next_action = np.argmax(q_table[next_discretized_state])
99         q_table[discretized_state][action] += learning_rate * (
100             reward + discount_factor * q_table[next_discretized_state][best_next_action] - q_table[discretized_state][action]
101         )
102
103         discretized_state = next_discretized_state
104         total_reward += reward
105
106         if terminated or truncated:
107             done = True
108
109         if done:
110             rewards.append(total_reward)
111             epsilon = max(epsilon_min, epsilon * epsilon_decay)
112
113             if (episode + 1) % 100 == 0:
114                 avg_reward = np.mean(rewards[-100:])
115                 print(f"Episode {episode + 1}, Averaging reward: {avg_reward:.2f}")
116
117             np.savetxt('q_table_mountaincar.npy', q_table)
118             print("Entrenamiento completado!")
119
120 # Publicación en RVIZ
121 rospy.init_node('mountaincar_visualization')
122 car_publisher = rospy.Publisher('/mountaincar/car_position', Marker, queue_size=10)
123 track_publisher = rospy.Publisher('/mountaincar/track', Marker, queue_size=10)
124
125 rate = rospy.Rate(30)
126
127 track_marker = create_track_marker()
128
129 state, _ = env.reset()
130 discretized_state = discretize_state(state)
131
132 done = False
133 while not rospy.is_shutdown() and not done:
134     action = np.argmax(q_table[discretized_state])
135     state, _, terminated, truncated, _ = env.step(action)
136     discretized_state = discretize_state(state)
137
138     # Publicar el marcador del carro
139     car_marker = create_car_marker([state[0]])
140     car_publisher.publish(car_marker)
141
142     # Publicar el marcador de la carretera en cada iteración
143     track_publisher.publish(track_marker)
144
145     if terminated or truncated:
146         done = True
147         rate.sleep()
148
149 env.close()
150

```

Código 17: Código Mountaincar_trained

Agente Pendulum con entrenamiento

Ejecución y resultados:

- En consola, se observan recompensas acumuladas que reflejan la capacidad del agente para estabilizarse.
- En RViz, se visualiza el movimiento y la estabilidad alcanzada por el péndulo.
- Consola:

```
Episodio 8700, Recompensa promedio: -232.58
Episodio 8800, Recompensa promedio: -208.67
Episodio 8900, Recompensa promedio: -233.54
Episodio 9000, Recompensa promedio: -241.98
Episodio 9100, Recompensa promedio: -229.90
Episodio 9200, Recompensa promedio: -270.18
Episodio 9300, Recompensa promedio: -220.47
Episodio 9400, Recompensa promedio: -244.65
Episodio 9500, Recompensa promedio: -267.67
Episodio 9600, Recompensa promedio: -234.21
Episodio 9700, Recompensa promedio: -260.62
Episodio 9800, Recompensa promedio: -232.47
Episodio 9900, Recompensa promedio: -249.59
Episodio 10000, Recompensa promedio: -232.63
Entrenamiento completado.
```

FOTO 13: Recompensas Pendulum_trained

- Código:

```

1 #!/usr/bin/env python3
2
3 import rospy
4 import gymnasium as gym
5 import numpy as np
6 from visualization_msgs.msg import Marker
7 from geometry_msgs.msg import Point
8
9 # Configuración de parámetros de Q-learning
10 action_space = (20, 20, 20) # Tres dimensiones: coseno, seno, velocidad angular
11 action_space = 5 # Cinco acciones para discretizar el espacio continuo de acciones en Pendulum
12 learning_rate = 0.01
13 discount_factor = 0.99
14 epsilon = 1.0
15 epsilon_decay = 0.995
16 epsilon_min = 0.01
17 num_episodes = 10000
18
19 # Inicialización de la tabla Q
20 q_table = np.zeros(state_space + (action_space,))
21
22 # Discretización del espacio de estados
23 def discretize_state(state):
24     bins = [
25         np.linspace(-1.0, 1.0, state_space[0]), # Coseno del ángulo
26         np.linspace(-1.0, 1.0, state_space[1]), # Seno del ángulo
27         np.linspace(-8.0, 8.0, state_space[2]) # Velocidad angular
28     ]
29     indices = [np.digitize(state[i], bins[i]) - 1 for i in range(len(state))]
30     return tuple(np.clip(indices, 0, np.array(state_space) - 1))
31
32 # Crear marcador para visualizar el péndulo
33 def create_pendulum_marker(theta):
34     marker = Marker()
35     marker.header.frame_id = "world"
36     marker.header.stamp = rospy.Time.now()
37     marker.ns = "pendulum"
38     marker.id = 0
39     marker.type = Marker.LINE_STRIP
40     marker.action = Marker.ADD
41     marker.scale.x = 0.05
42     marker.color.a = 1.0
43     marker.color.r = 1.0
44     marker.color.g = 0.0
45     marker.color.b = 0.0
46
47     pendulum_length = 1.0 # Longitud del péndulo
48     x = pendulum_length * np.sin(theta) # Coordenada X
49     z = pendulum_length * np.cos(theta) # Coordenada Z
50
51     base = Point(0.0, 0.0, 0.0)
52     end = Point(x, 0.0, z)
53
54     marker.points.append(base)
55     marker.points.append(end)
56
57     return marker
58
59 # Entrenamiento del agente
60 env = gym.make('Pendulum-v1', render_mode=None)
61 rewards = []
62
63 for episode in range(num_episodes):
64     state, _ = env.reset()
65     discretized_state = discretize_state(state)
66     total_reward = 0
67     done = False
68
69     while not done:
70         # Selección de acción (exploración/explotación)
71         if np.random.rand() < epsilon:
72             action_idx = np.random.randint(0, action_space)
73         else:
74             action_idx = np.argmax(q_table[discretized_state])
75
76         # Convertir acción discreta a continua
77         action = np.linspace(-2.0, 2.0, action_space)[action_idx]
78
79         # Ejecutar la acción
80         next_state, reward, terminated, truncated, _ = env.step(action)
81         next_discretized_state = discretize_state(next_state)
82
83         # Actualización de Q-table
84         best_next_action = np.argmax(q_table[next_discretized_state])
85         q_table[discretized_state][action_idx] += learning_rate * (
86             reward + discount_factor * q_table[next_discretized_state][best_next_action] - q_table[discretized_state][action_idx]
87         )
88
89         discretized_state = next_discretized_state
90         total_reward += reward
91
92         if terminated or truncated:
93             done = True
94
95         rewards.append(total_reward)
96         epsilon = max(epsilon_min, epsilon * epsilon_decay)
97
98         if (episode + 1) % 100 == 0:
99             avg_reward = np.mean(rewards[-100:])
100            print(f'Episodio {episode + 1}, Recompensa promedio: {avg_reward:.2f}')
101
102 np.save('q_table_pendulum.npy', q_table)
103 print("Entrenamiento completado.")
104
105 # Visualización en RViz
106 rospy.init_node('pendulum_visualization')
107 pendulum_publisher = rospy.Publisher('/pendulum/pendulum_marker', Marker, queue_size=10)
108 rate = rospy.Rate(10)
109
110 state, _ = env.reset()
111 discretized_state = discretize_state(state)
112
113 done = False
114 while not rospy.is_shutdown() and not done:
115     # Seleccionar acción entrenada
116     action_idx = np.argmax(q_table[discretized_state])
117     action = np.linspace(-2.0, 2.0, action_space)[action_idx]
118
119     # Ejecutar acción
120     next_state, _, terminated, truncated, _ = env.step(action)
121     discretized_state = discretize_state(next_state)
122
123     # Publicar marcador del péndulo
124     theta = np.arctan2(next_state[1], next_state[0]) # Ángulo del péndulo
125     pendulum_marker = create_pendulum_marker(theta)
126     pendulum_publisher.publish(pendulum_marker)
127
128     done = terminated or truncated
129     rate.sleep()
130
131 env.close()

```

Código 18: Código Pendulum_trained

Github de codigos entrenados:

<https://github.com/RousingManxx/AgentesOpenAIGymRviz>

Conclusión

La implementación de tres agentes distintos (Acrobot, MountainCar y Pendulum) utilizando GitHub Copilot como herramienta de asistencia en la programación ha proporcionado valiosas perspectivas sobre el potencial y las limitaciones de la inteligencia artificial generativa en el contexto educativo y de desarrollo.

Hallazgos Principales

Efectividad de GitHub Copilot

- La herramienta demostró capacidad para generar código base funcional para los tres entornos de OpenAI Gym.
- Se requirieron múltiples iteraciones y ajustes para obtener código completamente funcional.
- La calidad del código generado mejoró significativamente cuando se le proporcionaron ejemplos de referencia de implementaciones previas exitosas.

Proceso de Desarrollo

- El tiempo de desarrollo se redujo considerablemente al utilizar Copilot como asistente de programación.
- La herramienta mostró capacidad para integrar características adicionales cuando se le solicitaron (como visualización gráfica y monitoreo de recompensas).
- Se observó que Copilot aprende y adapta sus sugerencias basándose en el código existente en el proyecto.

Consideraciones Educativas

- GitHub Copilot demostró ser una herramienta útil para estudiantes que ya tienen conocimientos básicos de programación y comprenden los conceptos fundamentales de aprendizaje por refuerzo.
- La herramienta no elimina la necesidad de comprender los principios subyacentes del aprendizaje por refuerzo y la programación.

- Los estudiantes deben mantener un equilibrio entre el uso de la herramienta y el desarrollo de sus propias habilidades de resolución de problemas.

Implicaciones Pedagógicas

El uso de GitHub Copilot en entornos educativos debe ser considerado como una herramienta complementaria que:

- Acelera el proceso de implementación de soluciones conocidas
- Permite a los estudiantes concentrarse en conceptos de alto nivel
- Facilita la experimentación con diferentes aproximaciones a problemas complejos

Sin embargo, es crucial establecer pautas claras para su uso que:

- Fomenten el aprendizaje activo y la comprensión profunda
- Eviten la dependencia excesiva de la herramienta
- Promuevan la transparencia en el proceso de desarrollo

Recomendaciones

Para Educadores

- Integrar GitHub Copilot como parte de una estrategia de enseñanza más amplia
- Establecer objetivos claros de aprendizaje que vayan más allá de la simple generación de código
- Fomentar la reflexión crítica sobre las soluciones generadas automáticamente

Para Estudiantes

- Utilizar GitHub Copilot como una herramienta de apoyo, no como un sustituto del aprendizaje
- Dedicar tiempo a comprender el código generado y sus principios subyacentes
- Practicar la programación tanto con como sin la asistencia de la herramienta

Para Futuros Desarrollos

- Explorar la implementación de agentes más complejos utilizando GitHub Copilot
- Investigar la integración de otras herramientas de IA generativa en el proceso de desarrollo
- Documentar y compartir las mejores prácticas para el uso educativo de estas herramientas
- Este proyecto ha demostrado que GitHub Copilot puede ser una herramienta valiosa en el contexto educativo cuando se utiliza de manera apropiada y ética, manteniendo un equilibrio entre la eficiencia en el desarrollo y el proceso de aprendizaje.

Contexto y Consideraciones Éticas en el Uso de Inteligencia Artificial Generativa en la Educación

En el dinámico panorama de la programación actual, los generadores de código automáticos están emergiendo como herramientas alternativas para la implementación de tareas introductorias y la resolución de problemas conocidos. Este proyecto se centra en evaluar la confiabilidad de estas herramientas, específicamente GitHub Copilot, para encomendar tareas de programación a estudiantes principiantes en la resolución de problemas relativamente complejos. El objetivo es determinar si el código generado por estas herramientas es indetectable y en qué medida libera al estudiante de la responsabilidad de aprender las bases de la programación.

En este sentido, es crucial considerar las directrices y manifestaciones sobre el uso de la Inteligencia Artificial Generativa (IAGen) en entornos educativos. Según un documento reciente en el que está trabajando la Universidad Veracruzana, el uso de IAGen debe enmarcarse dentro de principios éticos y educativos claros que promuevan el uso consciente y responsable de estas tecnologías.

Consideraciones Institucionales y Comunitarias

- **Integración de IAGen en funciones sustantivas:** Emplear la IAGen en actividades de gestión, cursos académicos y proyectos de investigación para optimizar el trabajo de bajo nivel y ganar espacio para el pensamiento avanzado.
- **Promoción de la equidad en el acceso:** Incentivar el acceso de todas las personas a las tecnologías de IAGen y fomentar la responsabilidad algorítmica.
- **Capacitación y acompañamiento de la comunidad académica:** Desarrollar las capacidades de estudiantes y personal en el uso adecuado de IAGen mediante acciones de capacitación y actualización.

Consideraciones Académicas

- **Uso educativo de los modelos de lenguaje:** Utilizar los modelos de lenguaje amplio (LLM) para promover el aprendizaje y la comprensión de conceptos propios de las disciplinas académicas.
- **Respeto a las tradiciones disciplinarias:** Considerar la diversidad disciplinaria y las tradiciones de las comunidades académicas.
- **Honestidad intelectual y transparencia algorítmica:** Promover el uso ético de la tecnología y la transparencia en los procedimientos académicos y creativos.

Desarrollo de Soluciones IAGen

- **Diseño consciente de estrategias:** Colaborar con desarrolladores de IA para socializar las mejores prácticas y co-diseñar estrategias de soluciones académicas.
- **Preferencia por desarrollos abiertos y auditables:** Preferir algoritmos y metodologías de desarrollo que sean claros y transparentes en su procesamiento.

Estos principios subrayan la importancia de utilizar herramientas como GitHub Copilot de manera que no solo faciliten la programación, sino que también respeten los valores educativos y éticos fundamentales. El auge de la Inteligencia Artificial ha impulsado un desarrollo sin precedentes en el campo del procesamiento del lenguaje natural, y es esencial que su integración en la educación se realice de manera reflexiva y responsable.

Github en donde se encuentran los agentes utilizados en estos ejemplos:

https://github.com/RousingManxx/OpenAI_GYM-GithubCopilot

Referencias

- Acerca de GitHub Copilot - Documentación de GitHub. (s. f.). GitHub Docs. Recuperado 15 de julio de 2024, de <https://docs.github.com/es/copilot/about-github-copilot>
- (S/f). Openai.com. Recuperado el 13 de agosto de 2024, de <https://openai.com/index/openai-gym-beta/>
- Tozzi, C. (2024, 07 junio). *The past, present and future of AI coding tools*. App Architecture. <https://www.techtarget.com/searchapparchitecture/tip/366539577/The-past-present-and-future-of-AI-coding-tools>
- *AI code generation: Enhancing how developers code* | GitHub Resources. (s. f.). GitHub Resources. <https://resources.github.com/artificial-intelligence/what-is-ai-code-generation/>
- Maddison, L. (2023, 15 febrero). GitHub Copilot for business is here, using OpenAI to solve your programming worries. TechRadar. <https://www.techradar.com/news/github-copilot-for-business-is-here-with-openai-to-solve-your-programming-worries>
- Jain, S. (2024, 11 octubre). How to Install ROS on Mac Silicon M1/M2/M3 (Updated method). Medium. https://medium.com/@shubhjain_007/ros-on-mac-silicon-m1-m2-updated-method-e87086d84e45