Aprendizaje por Refuerzo (AR) con un enfoque de Q-Learning utilizando la estrategia de Epsilon-Greedy

Inicialización del Agente: Primero, importé las bibliotecas necesarias, como `gymnasium` para crear el entorno de simulación, `numpy` para hacer cálculos numéricos, y `pickle` para guardar o cargar datos de manera fácil. Después, construí una clase llamada `EpsilonGreedyQLearningAgent` que contenía la estructura completa de mi agente de aprendizaje. Configuré los parámetros claves:

- `alpha`, que era la tasa de aprendizaje, para controlar cuánto se ajustaban los valores aprendidos.
- `gamma`, el factor de descuento, para priorizar las recompensas a largo plazo.
- `epsilon`, que decidía la probabilidad de explorar acciones nuevas frente a aprovechar lo que ya conocía.

Creación de la Tabla Q: Al iniciar el agente, generé una tabla `Q` que almacenaba valores para cada combinación de estado-acción posible en el entorno. Esta tabla iba a servir como referencia para que el agente entendiera qué acciones le daban mejores recompensas en diferentes estados.
Interacción con el Entorno: Para que el agente pudiera aprender, lo hice jugar repetidamente en el entorno. En cada episodio:

- Seleccionaba una acción usando la estrategia Epsilon-Greedy, donde a veces el agente escogía la mejor acción conocida y, otras veces, exploraba opciones nuevas para descubrir más alternativas.
- Ejecutaba la acción, observaba el nuevo estado, y recibía una recompensa del entorno.

Actualización de la Tabla Q: Después de cada acción, aplicaba la fórmula de Q-Learning para actualizar la tabla Q, utilizando la recompensa obtenida y una estimación de las recompensas futuras. Así, la tabla Q se fue llenando de información útil, reflejando qué acciones llevaban a mejores recompensas en cada estado posible.
Ajuste de Epsilon: Después de cada episodio, iba reduciendo el valor de `epsilon` para que el agente explorara menos con el tiempo y se enfocara más en explotar el conocimiento adquirido. De esta manera, al avanzar en el entrenamiento, el agente tomaba decisiones basadas en lo que había aprendido, en lugar de probar cosas nuevas continuamente.
Guardado y Carga de la Tabla Q: Usé `pickle` para guardar y cargar la tabla Q, lo cual fue muy útil para poder detener y retomar el entrenamiento sin perder el conocimiento ya adquirido por el agente.
Visualización de Resultados: Al final, generé gráficos que mostraban cómo iban evolucionando las recompensas a lo largo de los episodios. Esto me permitió ver el progreso del agente y verificar que el entrenamiento estaba funcionando correctamente.

Aprendizaje por Refuerzo con Q-Learning, utilizando una estrategia de selección de acciones basada en Softmax

Inicialización del Agente: Primero, importé las bibliotecas necesarias, como `gymnasium` para crear el entorno, `numpy` para los cálculos numéricos, y `pickle` para guardar y cargar datos. Luego, definí la clase `SoftmaxQLearningAgent`, que contenía todos los componentes de mi agente. Configuré varios parámetros importantes:

- `alpha`, la tasa de aprendizaje, para ajustar cuánto peso daba a la nueva información.
- `gamma`, el factor de descuento, que daba prioridad a las recompensas futuras.
- `tau`, que actuaba como una "temperatura" en el algoritmo Softmax, controlando cuánto exploraba el agente.

Creación de la Tabla Q: Al crear el agente, también inicialicé una tabla `Q` con ceros. Esta tabla almacenaba valores para cada combinación de estado-acción en el entorno y me permitía llevar un registro de las recompensas esperadas por cada acción en cada estado.
Interacción con el Entorno: Para entrenar al agente, lo hice interactuar repetidamente con el entorno. En cada episodio:

- Seleccionaba una acción usando Softmax, una estrategia que selecciona acciones basándose en probabilidades, donde acciones con valores Q altos tenían más probabilidades de ser elegidas, pero aún había oportunidad de explorar otras acciones.
- Ejecutaba la acción elegida, observaba el nuevo estado y recibía una recompensa del entorno.

Actualización de la Tabla Q: Usaba la fórmula de Q-Learning para actualizar la tabla `Q` después de cada acción. Con esto, la tabla `Q` reflejaba el valor esperado de cada acción en cada estado, considerando las recompensas futuras.
Ajuste de Tau: Después de cada episodio, reducía el valor de `tau` para que el agente se centrara más en explotar el conocimiento aprendido en lugar de explorar continuamente. Esto ayudaba a que el agente se enfocara en las mejores acciones una vez que tenía una idea clara del entorno.
Guardado y Carga de la Tabla Q: Utilicé `pickle` para guardar y cargar la tabla `Q`, lo cual me permitió pausar y continuar el entrenamiento sin perder el conocimiento adquirido.
Visualización de Resultados: Al finalizar el entrenamiento, generé gráficos que mostraban cómo cambiaban las recompensas a lo largo de los episodios. Esto me permitió visualizar el progreso y confirmar que el agente estaba aprendiendo correctamente.

Aprendizaje por Refuerzo (AR) utilizando Q-Learning y una estrategia de selección de acciones basada en Upper Confidence Bound (UCB)

Inicialización del Agente: Primero, importé las bibliotecas esenciales, como `gymnasium` para el entorno de simulación, `numpy` para cálculos numéricos, y `pickle` para guardar y cargar la información del agente. Luego, definí la clase `UCBQLearningAgent`, donde establecí los componentes básicos de mi agente. Configuré los parámetros clave:

- `alpha`, o tasa de aprendizaje, que controlaba cuánto peso le daba a la nueva información.
- `gamma`, el factor de descuento, para priorizar recompensas futuras.
- `c`, un parámetro que definía la confianza en la selección de acciones, esencial en el método UCB.

Creación de la Tabla Q: Al iniciar el agente, inicialicé una tabla `Q` con ceros. Esta tabla `Q` almacenaba el valor de recompensa esperada para cada combinación de estado y acción. También creé un contador para llevar registro de cuántas veces se había tomado cada acción en cada estado, lo cual era necesario para el cálculo de UCB.
Interacción con el Entorno: Para entrenar al agente, lo hice interactuar con el entorno en múltiples episodios. En cada episodio:

- Elegía una acción usando el método UCB, que seleccionaba acciones con base en el valor `Q` y un término de exploración proporcional al parámetro `c`. Esto permitía un equilibrio entre explorar acciones menos probadas y explotar acciones que ya parecían prometedoras.
- Ejecutaba la acción seleccionada, observaba el estado resultante y recibía una recompensa del entorno.

Actualización de la Tabla Q: Después de cada acción, aplicaba la fórmula de Q-Learning para actualizar la tabla `Q`, integrando la recompensa obtenida y estimando las recompensas futuras. Esto permitía que la tabla `Q` reflejara el valor esperado de cada acción en cada estado.
Ajuste de `c`: Después de cada episodio, reducía el valor de `c` para ajustar el balance entre exploración y explotación a medida que el agente aprendía más sobre el entorno. Así, el agente podía enfocarse en las mejores decisiones conocidas una vez que había explorado lo suficiente.
Guardado y Carga de la Tabla Q: Utilicé `pickle` para guardar y cargar la tabla `Q` y el contador de acciones, lo cual me permitió pausar y retomar el entrenamiento sin perder el conocimiento adquirido.
Visualización de Resultados: Al final del entrenamiento, generé gráficos que mostraban el cambio de las recompensas a lo largo de los episodios, lo que me permitió evaluar el progreso y confirmar que el agente estaba aprendiendo de manera efectiva