Práctica 2

Reinforcement Learning

Sara Martín Aramburu

08/12/2023

Índice

1. Descripción de la práctica

La práctica consiste en modificar un entorno del juego típico de móvil *Snake* para lograr que tras entrenarlo con un modelo de la librería *Stable Baselines3,* se pueda lograr que el agente consiga jugar solo y hacerlo de manera correcta. La puntuación que se quiere obtener es de 30, es decir, conseguir que la serpiente se coma 30 manzanas.

La serpiente no puede chocar con su cuerpo ni con los bordes del mapa, ya que esto hace que se pierda la partida. Cuando la serpiente alcanza una manzana, su puntuación y longitud aumentan en 1.

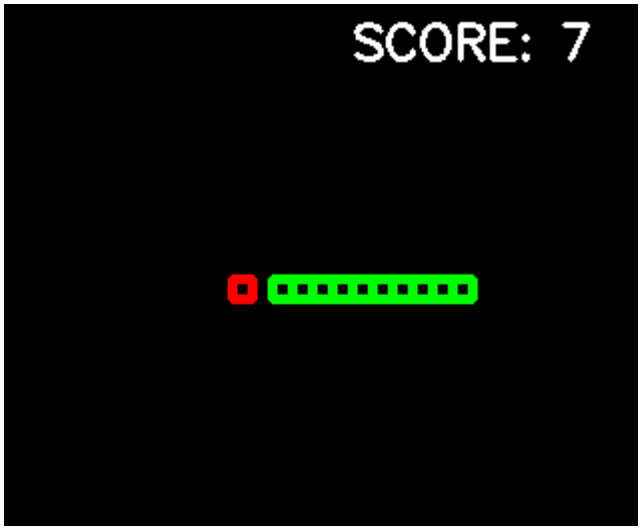

Figura 1: Imagen del juego Snake que se ha implementado.

El algoritmo que se ha utilizado para ello es Deep Q-Network (DQN), que consiste en entrenar una red neuronal para tomar las decisiones del juego.
En este entorno, hay 4 acciones posibles: avanzar hacia la izquierda, derecha, arriba o abajo. Esos son los 4 movimientos que el algoritmo deberá elegir.

2. Cambios realizados en el código

Para que el juego aprendiese de forma rápida y eficiente, se han realizado un par de cambios en el código.

2.1 Acciones

El primer cambio ha sido reducir el número de acciones de 4 a 3. Esto se debe a que muchas veces la serpiente retrocedía y, por ende, se chocaba con su cuerpo. Los nuevos movimientos son: girar a la izquierda, girar a la derecha y seguir hacia delante.

Es importante tener en cuenta que ahora hay que almacenar la dirección anterior de la serpiente para saber su nueva trayectoria. Según la acción, se calcula la nueva dirección, y sabiendo eso se mueve la serpiente.

Esto se calcula en la función *moveSnake*, que toma como argumento la acción que se quiere realizar.

Además, se ha incluído que la dirección inicial sea siempre hacia la derecha, y que la serpiente se encuentre al inicio en la mitad del mapa.

2.2 Recompensas

Para que el agente sepa qué es lo que tiene que hacer en el entorno, es importante definir una función de recompensa que se ajuste al problema. Tras muchos intentos y muchas pruebas de diferentes funciones de recompensa, se ha decidido que la serpiente obtenga -10 puntos cuanto se choca con las paredes o con su propio cuerpo, y obtiene una recompensa de 10 puntos cada vez que se come una manzana.

Lo que se consigue con este es que el objetivo de la serpiente sea alcanzar las manzanas, ya que si se queda dando vueltas por el mapa no obtiene recompensa, y si se choca obtiene una recompensa negativa, así que la única forma de conseguir recompensa positiva es esa, comer manzanas.

2.3 Observaciones

El último cambio que se ha realizado es sobre la observación. Se ha decidido pasarle más información para que el algoritmo elija la acción más acertada. Se han añadido las siguientes cuestiones:

- La dirección actual de la serpiente.
- Si va a chocar al tomar cada una de las direcciones posibles.
- La posición de la manzana con respecto a la cabeza, para saber si va bien encaminado y se está acercando a su objetivo.

Toda esta información se incluye en forma de booleanos, y la observación inicial no contiene información.

3. Algoritmos y evaluaciones

Se han utilizado 3 algoritmos diferentes que se han comparado: DQN, A2C y PPO.

Estos algoritmos han sido aportados por la librería *stable baselines 3*, ya que nos aporta diferentes algoritmos eficientes y facilita la creación y entrenamiento de modelos en diferentes entornos. Todos ellos se han entrenado con 170000 *steps*, y se han obtenido las gráficas correspondientes a los resultados.

Hay que destacar que con ese número de *steps*, utilizando el algoritmo A2C se quedaba en un bucle infinito en el que la serpiente solo daba vueltas, así que se ha puesto la recompensa media y desviación estándar a 0.

En la primera, se observa la recompensa media de cada uno de los modelos, mientras que en la segunda se muestra la desviación estándar de la recompensa en cada uno de ellos.

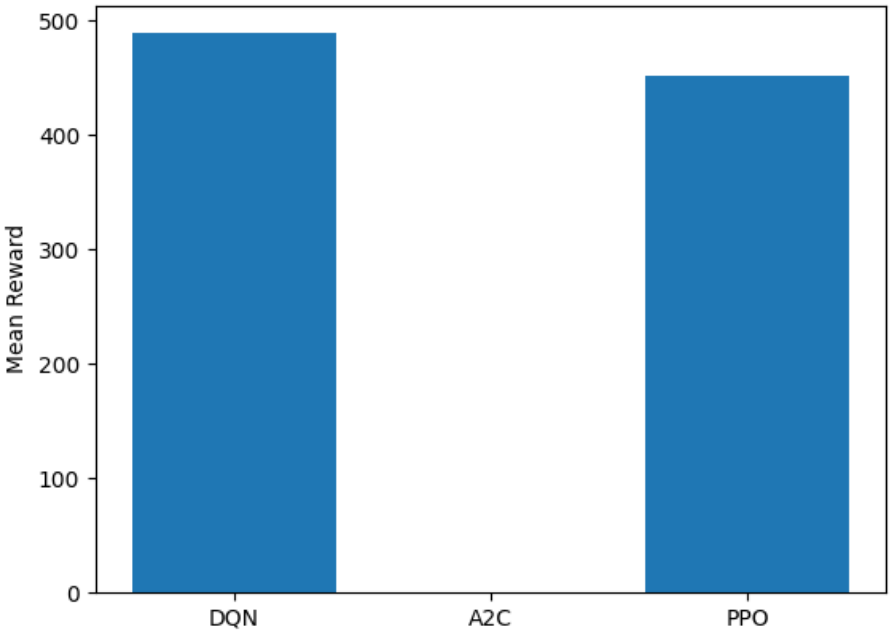

Figura 2: Recompensa media utilizando cada uno de los algoritmos.

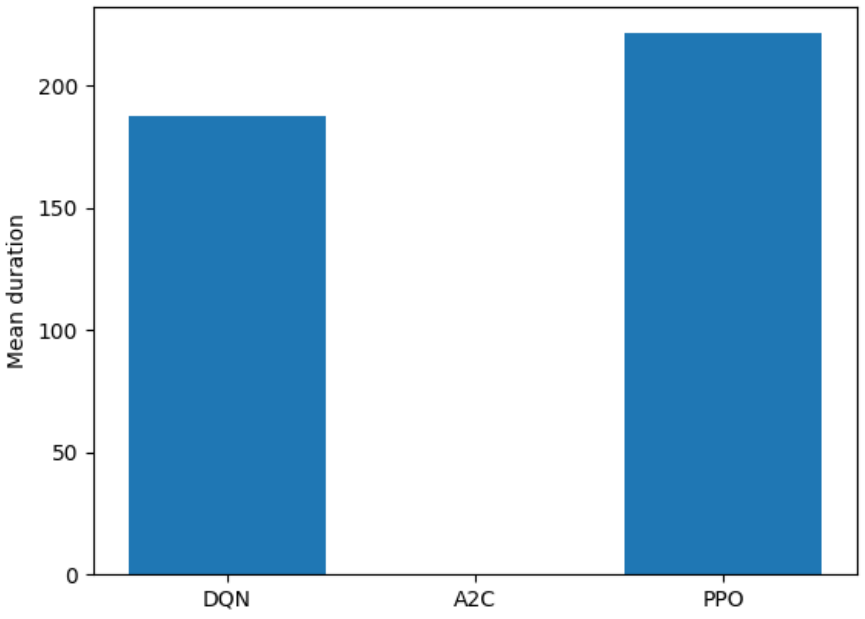

Figura 3: Desviación estándar utilizando cada algoritmo.

Es por eso por lo que la conclusión es que el mejor algoritmo para este problema es DQN, ya que es el que obtiene mejor medida de recompensa. Además, su desviación estándar es menor, por lo que es más estable. De todas formas, las métricas pueden ir variando en cada

una de las ejecuciones, por lo que lo ideal sería realizar muchas más pruebas y obtener mayores conclusiones.

4. Problemas y dificultades

El planteamiento inicial fue buscar una función de recompensa adecuada para que la serpiente supiera directamente hacia donde tenía que ir. Se probaron diferentes cosas, como calcular la distancia euclídea y de Manhattan a la manzana, ofrecer una recompensa si se acercaba a ella, aumentar la recompensa según el número de manzanas que se había comido, penalizar los movimientos innecesarios… Pero nada de eso dió resultado.

Además, probando esas funciones de recompensa se observaron diferentes comportamientos curiosos. Inicialmente, la serpiente aprendía a no chocar con su cuerpo, pero no lograba llegar a la manzana. Tras dejarla entrenando un rato más, aprendía a no chocar con la pared, aunque tampoco lograba su objetivo.

Cuando se incluyó una recompensa positiva por acercarse a la manzana, la serpiente optaba por seguir dando vueltas en lugar de comerse la manzana, ya que eso le ofrecía una mayor recompensa.

Por otro lado, hubo varios problemas a la hora de calcular la dirección que tenía que seguir la serpiente con la nueva acción, aunque esto se logró solventar de forma relativamente sencilla.

Asimismo, se decidió quitar el render para entrenar ya que tardaba mucho más y realmente no era necesario. En un principio se quería ver cómo el agente iba aprendiendo, pero no fue viable y solo se visualizó el resultado final, en el que se veía cómo jugaba después de haber entrenado y si lograba o no su objetivo.