Universidad ORT Uruguay

Obligatorio

Inteligencia Artificial

Marcelo Pérez Jodal (227362)

Federico Iglesias (244831)

Francisco Rossi (219401)

2022

Índice:

1. Resumen del abordaje de la tarea

Cartpole

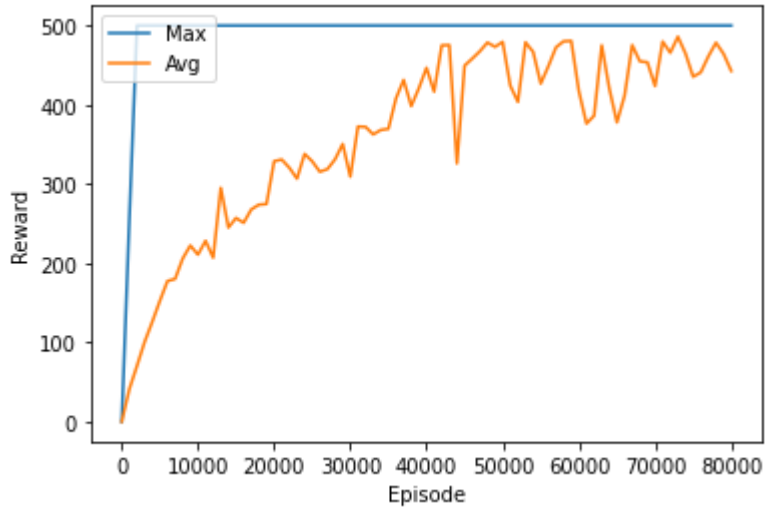
La idea fue trabajar en base al algortimo Q-learning para poder crear un simulador capaz de soportar un aterrizaje en Marte de forma completamente automática. Para esto nos basamos en el ambiente de simulación creado por gym Cartpole para crear nuestro propio simulador.

Para entrar más en detalle sobre el asunto queremos mencionar que significa Q-learning. Se trata de un algoritmo que permite aprender en base a la interacción del agente. Dado esta situación queremos que el agente interactúe con un simulador y aprenda, antes de que haga la interacción con el sistema de aterrizaje real. Lo que trata de hacer es estimar una policy óptima, que le indique al agente cómo actuar frente a las distintas situaciones que se puede llegar a encontrar interactuando con el sistema.

Un enfoque que toma este algoritmo es el de no siempre recurrir al mismo método para obtener una acción, sino que considera la exploración de forma estocástica, con el fin de descubrir alguna nueva acción para determinado estado y así intentar mejorar su aprendizaje.

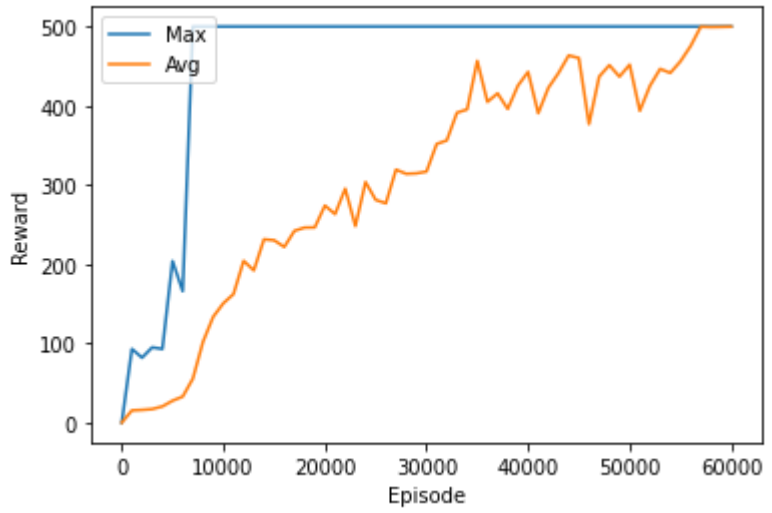
Nuestra primera implementación del algoritmo interactuaba con nuestro simulador obteniendo malos resultados. Pudimos encontrar varios problemas relacionados a esto. Lo primero que decidimos cambiar fue la tabla del algoritmo, encargada de ir guardando los valores para cada estado, ya que este contenía valores discretos y el ambiente proporciona elementos continuos. Se adaptó con una herramienta de numpy (lindspaces) para adaptar el espacio de valores continuos a un grupo discreto. También sirvió para ajustar los límites de valores posibles ya que según la documentación, cuando se sobrepasa algunos valores específicos en su posición, velocidad, ángulo o velocidad angular el sistema falla en mantener su objetivo en pie.

Luego de algunos otros cambios en nuestro código, como por ejemplo cómo calcular la acción de forma aleatoria según un epsilon, nuestro algoritmo mejoró. Según la documentación de gym, el objetivo de las recompensas para la versión 1 del simulador es de 475 pasos sin dejar caer el objetivo. Nuestro agente podía alcanzar rápidamente en algún episodio alcanzar los 500 pasos, siendo este el máximo, pero en promedio nunca superaba los 400 pasos sin antes dejar caer el palo.
Se observó que esto fue debido a los parámetros que habíamos seleccionado. Nuestro epsilon, era demasiado grande por lo que al explorar demasiado nunca llegaba en promedio a 500 pasos sin dejarlo caer como se puede observar en la siguiente imagen. Aproximadamente 1 hora corriendo



[1]

Lo que se logró ajustando esto parámetros como por ejemplo una vez superado el umbral de la documentación (475 pasos) es indicarle que no explore más, sino que utilice el “exploit” de lo que ya se conoce en la Q-table.



[2]

Esto es un resultado a comprar con el anterior luego de realizarle las modificaciones, donde se puede observar que logra alcanzar a los 500 pasos en promedio pero también lo logra en menos intentos o episodios. Luego de resolverlo 5 veces en promedio, decidimos cortarlo el simulador porque entendemos que no aporta ningún valor adicional. Cabe destacar que el tiempo de entrenamiento se reduzco notariamente a cerca de la mitad adjustando parámetros, donde el algoritmo final alcanza su objetivo final en alrededor de 40 minutos, dependiendo la máquina. Otro cambio que influyó bastante fue el actualizar la Q-table sin importar si el ambiente había determinado como done o no, que antes solo lo hacíamos en el caso de que no haya terminado el episodio. Esto fue clave para mejorar ya que es verdad que tmabién

[1] Gáfica encontrada en Documentation/CartPole/80kRun.png
[2] Gáfica encontrada en Documentation/CartPole/60kRun.png

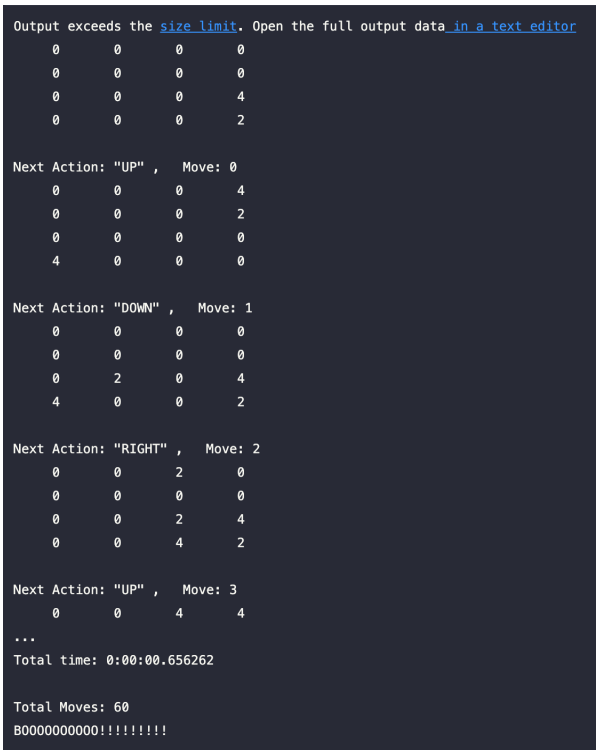
estamos actualizando la tabla cuando el palo se cae, pero también cuando en alguna corrida llega a 500 pasos, siendo este el objetivo.

Parámetros óptimos encontrados:

- Epsilon: 0.06, luego 0 si supera el umbral en promedio
- Episodios: >80000
- Gamma: 0.0995
- Learning Rate (lr): 0.14

2048

Todos los integrantes del equipo teníamos experiencia previa con el juego 2048 y conocíamos algunas estrategias para jugar prosperamente, como intentar acumular fichas de valor alto en una esquina.



La primera interacción con esta tarea fue de prueba con el simulador GameBoard. El Random Agent previsto permitió correr simulaciones con pasos aleatorios y así comenzar a conocer la naturaleza del problema, auxiliados por la función de board.render() para la visualización del tablero. Los primeros resultados fueron de un alcance muy limitado, con un tiempo total de 0:00:00.656262 y 60 movidas hasta perder.

Se evaluó la posibilidad de usar las técnicas Minimax y Expectimax, y se decidió por la última dada la mejor adecuación al ambiente. El GameBoard agrega fichas de forma aleatoria, como se puede ver en la implementación de su función: __add_random_tile. Tanto Minimax como Expectimax buscan la acción que maximice los resultados del agente, pero Minimax espera del oponente la acción que minimice su retorno, mientras que Expectimax espera una acción estocástica de su oponente. Por tanto, Expectimax es la técnica adecuada para el problema.

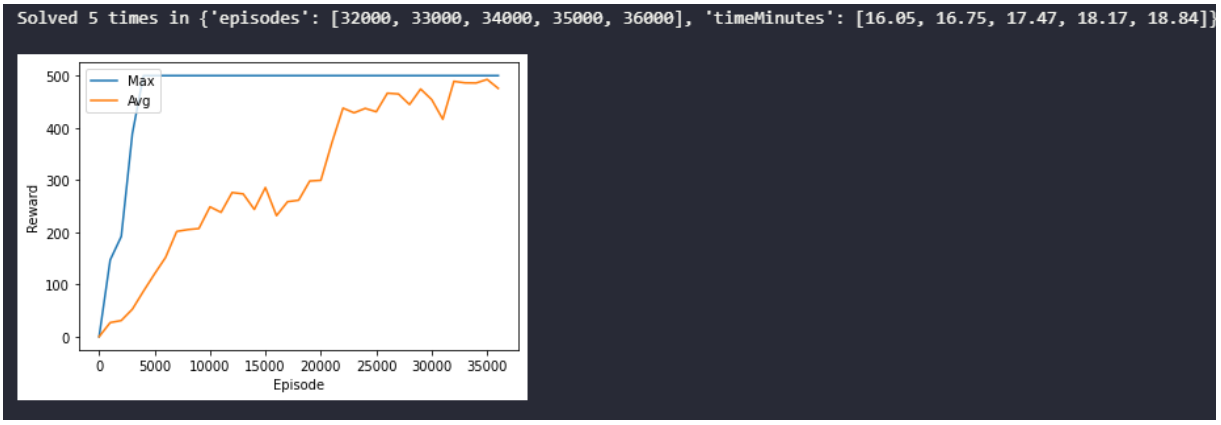
Luego, se desarrolló la clase ExpectimaxAgent, que implementa la interfaz Agent y por tanto define las funciones de play y heuristic_utility. TENEMOS QUE EXPLICAR LA HEURÍSTICA.

2. Desempeño de sus soluciones

CartPole

A pesar de contar con tiempos bastantes buenos para resolver el problema (cerca de 40 min), decidimos alcanzar niveles de optimización superiores. Para esto se realizó una investigación sobre el código y pudimos observar que al menos una vez cada 1000 episodios se alcanzaban los 500 pasos cumpliendo el objetivo final. Esto quiere decir que se registraron los correctos pasos a seguir en la tabla al menos una vez. Este simple hecho nos hizo darnos cuenta que si limitábamos el explorar aún más, podríamos obtener mejores resultados en cuanto a performance. Luego de varias corridas probando distintos formas de limitar el epsilon, llegamos a la consluión de que si en alguna corrida alcanzó los 500 pasos, el epsilon debía ser reducido cercano a 0 y si en ninguna corrida se lograba el objetivo se volvía a setear el epsilon para que pdiera seguir explorando en algunos casos.

Los resultados son los siguientes:



[3]

El algortimos resolvió el problema superando los 475 pasos que indica la documentación, 5 veces seguida en la mitad del tiempo, el más rápido en 16 minutos. Pero al limitar tanto la exploración, se puede observar en la gráfica que nunca llega a 500 pasos exatamente pero si muy cercano a este valor (492).

[3] Gáfica encontrada en Documentation/CartPole/30kRun.png

2048

El agente Expectimax implementado logró ganar el juego en todos las pruebas realizadas, a continuación analizamos una como ejemplo:

El tiempo total de ejecución fue 0:01:53.311548, menor a los dos minutos, y logró completar 946 acciones.

```
···  Output exceeds the size limit. Open the full output data in a text editor
          0       0       2       2
          0       0       0       0
          0       0       0       0
          0       0       0       0

     Next Action: "RIGHT" ,   Move: 0
          0       0       2       4
          0       0       0       0
          0       0       0       0
          0       0       0       0

     Next Action: "DOWN" ,   Move: 1
          0       0       0       0
          0       0       0       0
          0       0       2       0
          0       0       2       4

     Next Action: "UP" ,   Move: 2
          0       0       4       4
          0       0       0       0
          0       0       0       0
          2       0       0       0

     Next Action: "RIGHT" ,   Move: 3
          0       0       0       8
     ...
     Total time: 0:01:53.311548

     Total Moves: 946
     WON THE GAME!!!!!!!
```

3. Notas de advertencia