

Obligatorio Inteligencia Artificial

Paula Cantera - 256680

Julio 2024

M7A

Los archivos se encuentran en el siguiente repositorio de github:

https://github.com/paulacantera/IA_Obl_256680

Parte 1

Taxi

El objetivo de taxi es levantar pasajeros y dejarlos en el destino en la mínima cantidad de movimientos posibles. Se usó Q learning para tratar de encontrar la mejor posible acción siguiente dado el estado en el que estamos. Con el objetivo de maximizar la reward decide que acciones tomar. El entorno es discreto con estados definidos por la ubicación del taxi, la ubicación del pasajero y la ubicación del destino.

Parámetros Utilizados

Alpha : 0.1 (tasa de aprendizaje): Alpha controla cuánto peso se le da a la nueva información en comparación con la información existente. Un valor de $\alpha = 0.1$ significa que el 10% de la actualización de la tabla Q se basa en la nueva información, mientras que el 90% se basa en la información ya existente. Esto permite que el aprendizaje sea más estable, ya que no se sobreescriben completamente las estimaciones anteriores con cada nueva actualización.

Gamma : 0.95 (factor de descuento): Gamma determina la importancia de las recompensas futuras. Un valor de $\gamma = 0.95$ significa que se considera el 95% del valor de las recompensas futuras al actualizar la tabla Q. Esto incentiva al agente a buscar recompensas a largo plazo en lugar de recompensas inmediatas, promoviendo decisiones que maximicen la recompensa total acumulada a lo largo del tiempo.

Epsilon (ϵ): 1.0 (tasa de exploración inicial): Epsilon define la probabilidad de que el agente elija una acción aleatoria en lugar de la acción óptima según la tabla Q actual. Un valor inicial de $\epsilon = 1.0$ significa que el agente comienza explorando todas las posibles acciones con probabilidad igual, lo que es crucial para asegurarse de que el agente explore el entorno y evite caer en un óptimo local.

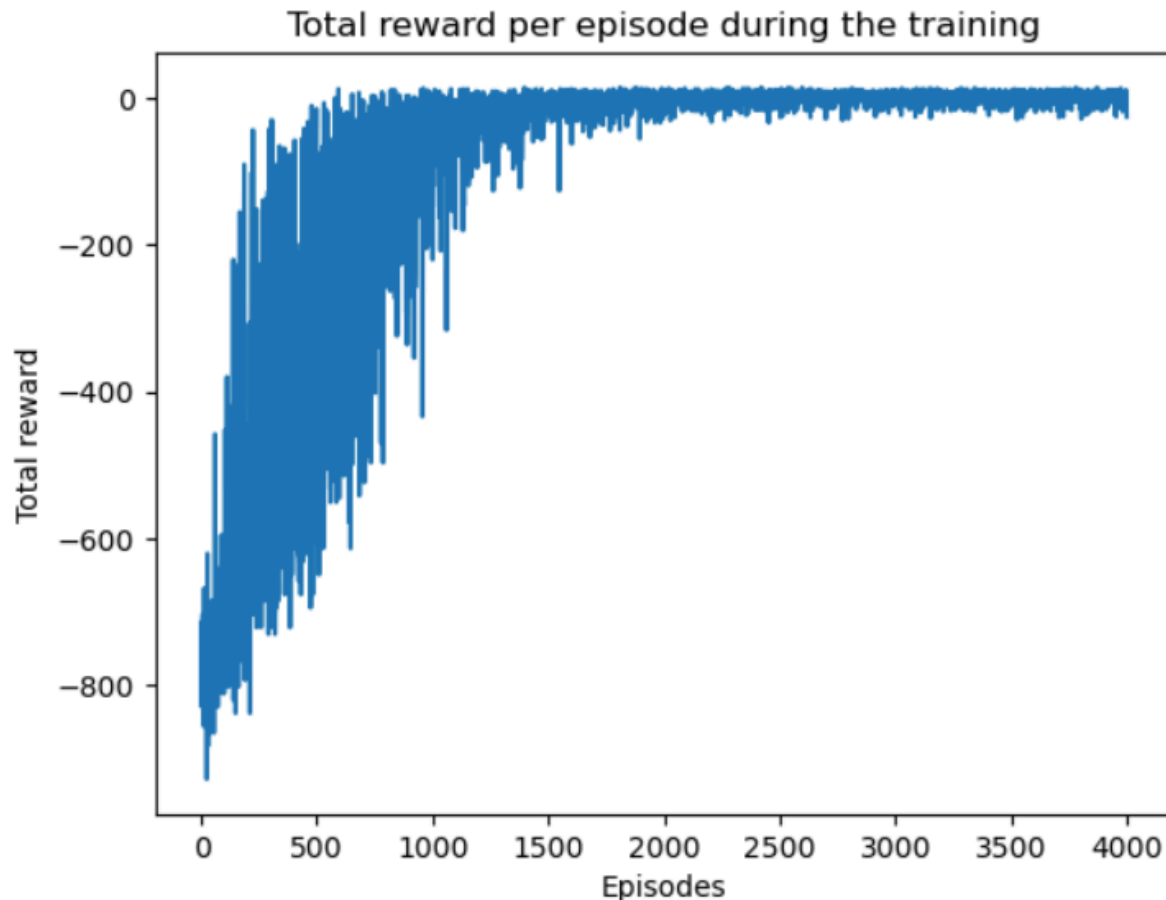
Este valor de epsilon me trajo problemas al principio porque el agente tenía la misma tasa de exploración durante todo el entrenamiento, lo cual no tiene sentido porque lo que queremos es que a medida de que se entrena explore menos y explote más. Por esta razón se hicieron dos nuevos parámetros: epsilon_i y epsilon_f. Epsilon comienza con un valor de 1.0 y se disminuye dentro de la función de Q learning gradualmente a 0.1 usando un factor de decaimiento.

Este cambio hizo cambiar el reward de -112 a una razonable

La cantidad de episodios necesarios para el entrenamiento fueron 4000. Al principio se utilizaron 10000 pero viendo la gráfica "total reward per episode during the training" me di cuenta que los valores luego de cierta cantidad de episodios eran constantes entonces se decidió disminuirlo de a poco mientras se obtenía la misma reward. Con esta cantidad de episodios el entrenamiento demora unos 3 segundos.

Después de completar el entrenamiento, se evaluó el rendimiento del agente ejecutando varios episodios y registrando las recompensas obtenidas con la función de evaluación de la política.

El gráfico de recompensas por episodio durante el entrenamiento muestra cómo el agente mejora con el tiempo. Al principio, las recompensas son bajas debido a la alta exploración, pero a medida que epsilon disminuye, el agente explora menos y explota más la política aprendida, resultando en recompensas más altas.



Se obtuvo una recompensa promedio de alrededor de 7 a 9 después de 100 episodios evaluando la política, indicando que el agente aprendió a minimizar los movimientos y maximizar las entregas exitosas de pasajeros. Esta recompensa tiene sentido evaluando el environment de taxi.

PENDULO

El objetivo de esta parte fue implementar y entrenar un agente capaz de balancear el péndulo usando técnicas de reinforcement learning, en este caso Q-learning.

Las observaciones proporcionadas son un vector continuo $[x, y, \text{velocity}]$, donde x y y representan la posición del péndulo en coordenadas cartesianas y velocity es la velocidad angular.

Las acciones son continuas, representando el toque aplicado al péndulo. Para mejorar la recompensa intente cambiar la discretización varias veces pero la que mejor reward me dio fue en 10 valores.

A diferencia de el environment de taxi, el cual es discreto y finito, con un número limitado de posiciones y estados definidos, el péndulo fue un desafío más grande. En este caso, el environment es continuo y potencialmente infinito, requiere discretización de los estados para aplicar Q-learning.

Al principio costó entender como se componía la tabla Q, pero luego se entendió que es de 4 dimensiones debido a la discretización de tres observaciones y acciones continuas.

El desafío más grande en la implementación fue entender y adaptarse a la implementación con entornos continuos.

Para la actualización de la tabla Q, Se calcula el índice de la acción y el estado siguiente, Se encuentra la mejor acción para el estado siguiente, Se calcula el objetivo de actualización temporal (TD target), Se actualiza el valor Q para la combinación actual de estado y acción

Parámetros de Q-Learning:

Tasa de aprendizaje : 0.1

Factor de descuento : 0.99

Epsilon inicial : 1.0

Epsilon final : 0.1

Decaimiento de epsilon: 0.999

Número de episodios: 50000

Al igual que en el entorno de taxi, al principio solo se tenía un epsilon, que valía 1.0 y cambiar a tener uno inicial y uno final cambio drásticamente los resultados. La razón la se explico anteriormente pero básicamente es por la necesidad de explorar más al principio y la necesidad de explotar más al final.

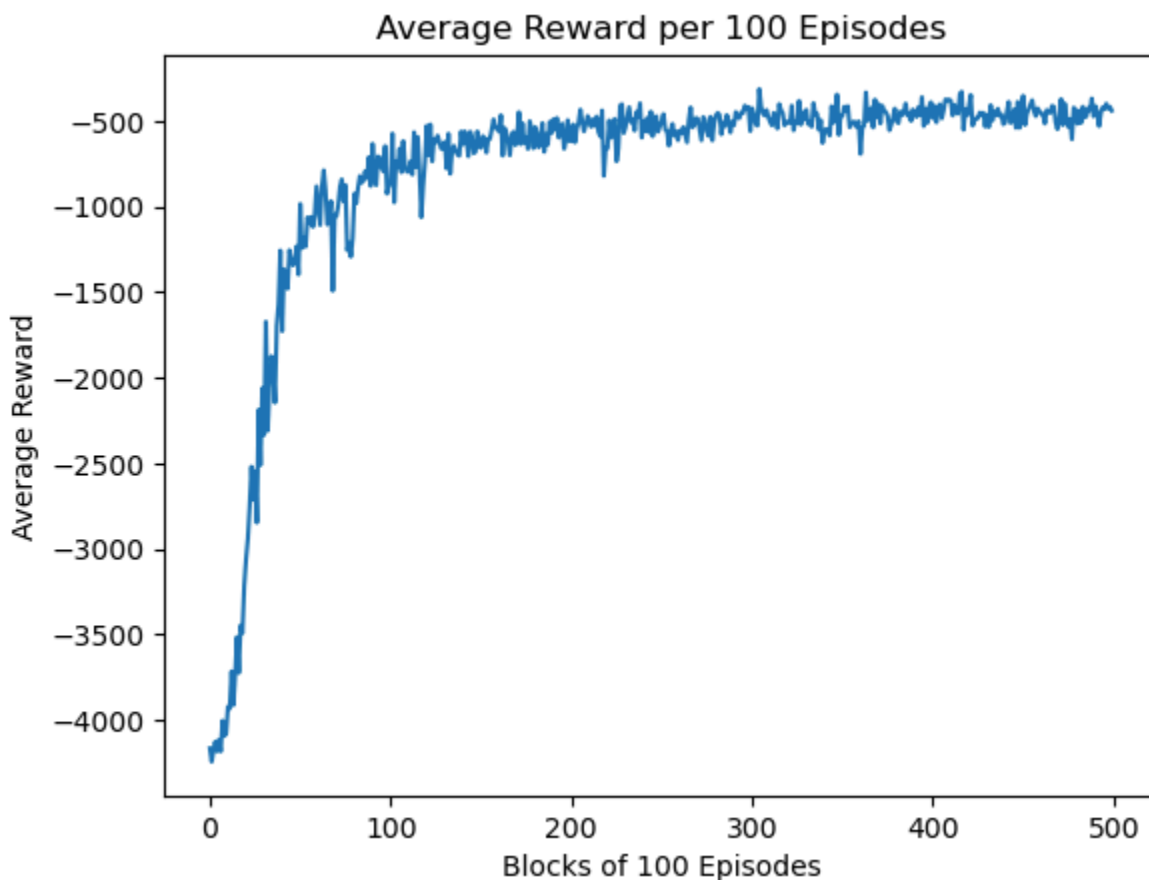
Se evaluó la política aprendida sobre 100 episodios para medir la efectividad del agente en el entorno de prueba con la función de evaluar policy.

El tiempo de ejecución promedio de 60000 episodios es de una hora.

La diferencia abismal de tiempo de ejecución con el contexto de taxi se debe a que este es lento en converger, porque es complejo y continuo.

La recompensa promedio al principio fue muy baja, mejorando gradualmente hasta alcanzar un promedio de aproximadamente -176.9 en los últimos 100 episodios.

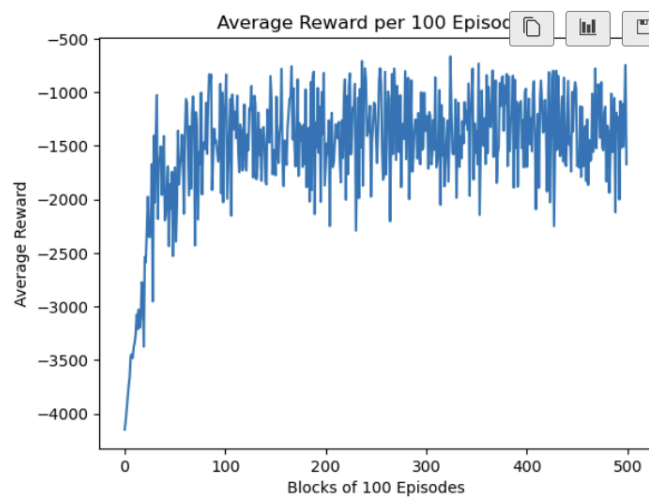
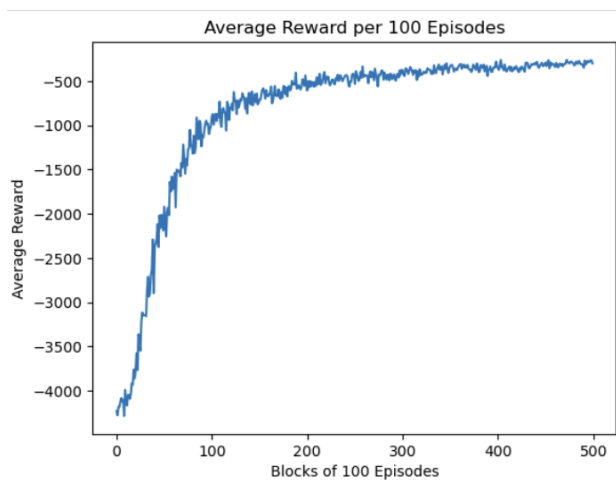
Se creó un gráfico que muestra la recompensa promedio obtenida por cada bloque de 100 episodios durante el proceso de entrenamiento. Este gráfico demuestra cómo la recompensa mejora con el tiempo a medida que el agente aprende a controlar mejor el péndulo.



El uso de Q-Learning en el entorno del péndulo invertido mostró mejoras graduales y consistentes en la recompensa acumulada, indicando que el agente pudo aprender a balancear el péndulo de manera más efectiva a medida que avanzaba el entrenamiento.

La reward obtenida no es la optima pero se acerca mucho a ella. Luego de implementarlo se investigó y quizás haber implementado otras técnicas avanzadas como DDPG podrían haber proporcionado mejor rendimiento y eficiencia.

Los primeros gráficos de la average reward probando distintas discretizaciones de los estados:



Coin Game

Las técnicas evaluadas fueron Minimax y Expectimax. Además, se exploró la optimización de parámetros de la heurística para mejorar el rendimiento del agente.

Al principio costó el entendimiento del funcionamiento del juego entonces se aplicó expectimax, pero este tipo de agentes se utiliza para juegos estocásticos.

Minimax terminó siendo la técnica utilizada porque coin game es un juego determinístico y de suma cero, donde el agente UONI juega de manera óptima (es experto en el juego).

La primera implementación de minimax fue muy básica, con una heurística que solo evaluaba en base a la cantidad de coins restantes en el tablero y sin determinar una max depth.

La primera ejecución se tuvo que cortar a las 3 horas porque minimax estaba explorando todos los movimientos posibles hasta el final del juego, entonces se limitó la profundidad de la búsqueda definiendo un `max_depth = 3`.

Primero se implementó una heurística que evaluaba en base a la cantidad de coins restantes en el tablero. Luego esta se fue actualizando y mejorando.

La heurística elegida combina el número total de las monedas restantes y el número de secuencias de monedas en el tablero. Se utilizaron las constantes a y b que ajustan la importancia relativa de cada componente.

Para la elección de a y b se utilizó Monte Carlo: se generaron aleatoriamente varios conjuntos de parámetros (entre -3 y 3), se evaluaron múltiples partidas contra UONI y luego se seleccionaron los mejores parámetros para iteraciones adicionales hasta converger a un conjunto óptimo.

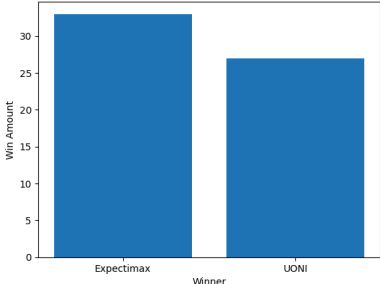
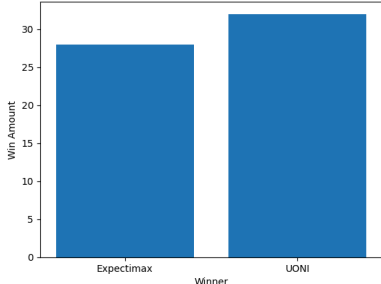
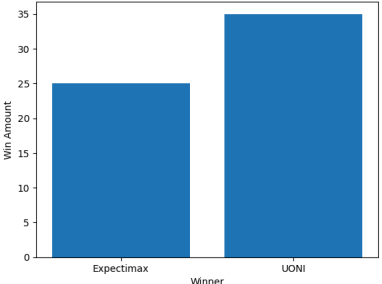
Estos parámetros se generaron considerando que comienza jugando el minimax agent y que el juego es nivel: "medium". Por lo tanto que los valores van a maximizar las ganancias para ese contexto.

Para el número de secuencias de monedas en el tablero, la constante que lo multiplica dio -2.25, lo que indica que pesa negativamente.

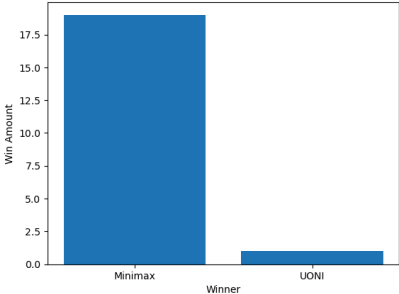
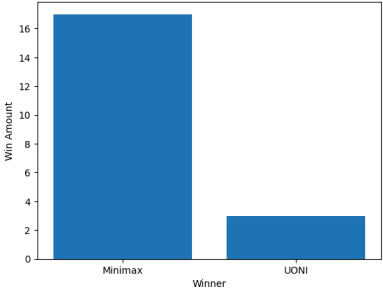
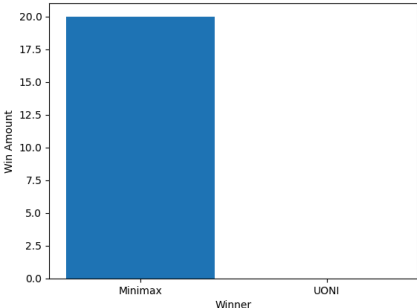
Para la cantidad de monedas en el tablero la constante dio 2.27, lo que significa que se van a tomar favorables los estados con más cantidad de monedas restantes en el tablero.

Se realizaron pruebas para evaluar el rendimiento de los agentes contra el agente UONI en diferentes niveles de dificultad.

Expectimax vs UONI:
(el de la izquierda es Expectimax y el de la derecha UONI)

Nivel	Easy Level	Medium Level	Hard Level																		
Grafico	<div><p>Win Amount per Winner at easy Level</p><table><thead><tr><th>Algorithm</th><th>Win Amount</th></tr></thead><tbody><tr><td>Expectimax</td><td>33</td></tr><tr><td>UONI</td><td>27</td></tr></tbody></table></div>	Algorithm	Win Amount	Expectimax	33	UONI	27	<div><p>Win Amount per Winner at medium Level</p><table><thead><tr><th>Algorithm</th><th>Win Amount</th></tr></thead><tbody><tr><td>Expectimax</td><td>28</td></tr><tr><td>UONI</td><td>32</td></tr></tbody></table></div>	Algorithm	Win Amount	Expectimax	28	UONI	32	<div><p>Win Amount per Winner at hard Level</p><table><thead><tr><th>Algorithm</th><th>Win Amount</th></tr></thead><tbody><tr><td>Expectimax</td><td>25</td></tr><tr><td>UONI</td><td>35</td></tr></tbody></table></div>	Algorithm	Win Amount	Expectimax	25	UONI	35
Algorithm	Win Amount																				
Expectimax	33																				
UONI	27																				
Algorithm	Win Amount																				
Expectimax	28																				
UONI	32																				
Algorithm	Win Amount																				
Expectimax	25																				
UONI	35																				
Tiempo de ejecucion (minutos)	4	4	6																		

Minimaxvs UONI:
(el de la izquierda es Minimax y el de la derecha UONI)

Nivel	Easy Level	Medium Level	Hard Level																		
Grafico	<div><p>Win Amount per Winner at easy Level</p><table border="1"><thead><tr><th>Winner</th><th>Win Amount</th></tr></thead><tbody><tr><td>Minimax</td><td>18.5</td></tr><tr><td>UONI</td><td>1.0</td></tr></tbody></table></div>	Winner	Win Amount	Minimax	18.5	UONI	1.0	<div><p>Win Amount per Winner at medium Level</p><table border="1"><thead><tr><th>Winner</th><th>Win Amount</th></tr></thead><tbody><tr><td>Minimax</td><td>16.5</td></tr><tr><td>UONI</td><td>3.0</td></tr></tbody></table></div>	Winner	Win Amount	Minimax	16.5	UONI	3.0	<div><p>Win Amount per Winner at hard Level</p><table border="1"><thead><tr><th>Winner</th><th>Win Amount</th></tr></thead><tbody><tr><td>Minimax</td><td>20.0</td></tr><tr><td>UONI</td><td>0.0</td></tr></tbody></table></div>	Winner	Win Amount	Minimax	20.0	UONI	0.0
Winner	Win Amount																				
Minimax	18.5																				
UONI	1.0																				
Winner	Win Amount																				
Minimax	16.5																				
UONI	3.0																				
Winner	Win Amount																				
Minimax	20.0																				
UONI	0.0																				
Tiempo de ejecucion (minutos)	2	2	2																		

Comparando los resultados y el tiempo de ejecución puedo confirmar mi hipótesis: minimax es la estrategia adecuada para el Coin Game.

Los resultados indican que el agente Minimax, con una heurística bien optimizada, puede competir eficazmente contra UONI incluso contra niveles de dificultad altos.

Parte 2

Alpha-beta pruning

Esta tecnica mejora la eficiencia de el algoritmo de busqueda MInimax al eliminar ramas del arbol de busqueda que no afecten a la desicion final. Esto se logra menteniendo dos valores: alpha y beta, que representan los limites inferiores y superiores para la evaluacion de los nodos.

Se utilizo la logica de la Figura 3 del paper para agregarla en lo que era la funcion de minimax implementada anteriormente, es decir, durante la búsqueda, se actualizan los valores de alpha y beta y se utilizan estos valores para podar ramas innecesarias del árbol de búsqueda. Tal cual como dice en el paper, la primer llamada del algoritmo es con $\alpha = -\infty$ y $\beta = \infty$.

Si en algún momento, alpha es mayor o igual a beta, podemos detener la evaluación de ese nodo (poda).

Al evaluar este agente

El de la izquierda es el agente alpha beta y el de la derecha UONI

Nivel	Easy Level	Medium Level	Hard Level
Grafico			
Tiempo de ejecucion (minutos)	00:14	00:17	00:18

Estos resultados reflejan que el tiempo de ejecucion bajo muchisimo (por la poda) pero la eficiencia a la hora de ganar no. Esto se podria dar al tipo de heuristica o a estar podando de manera incorrecta.

