Aprendizaje Automático

GRADO EN INGENIERÍA INFORMÁTICA

Práctica 2: Aprendizaje por Refuerzo

Curso 2020/2021

Jorge Rodríguez Fraile, 100405951, Grupo 83, 100405951@alumnos.uc3m.es
Carlos Rubio Olivares, 100405834, Grupo 83, 100405834@alumnos.uc3m.es

Índice

Introducción

En esta práctica, intentaremos hacer un agente de Pac-man que pueda resolver mapas mediante aprendizaje por refuerzo. Intentaremos hacer que esto sea un proceso cuantitativo, es decir, primero, haremos que funcione bien en mapas sencillos, sin mucha dificultad, para después ir ampliando su efectividad mediante el cambio de las funciones de refuerzo, o de atributos. En definitiva, creemos que conseguiremos un mejor resultado que en prácticas anteriores ya que esta técnica está basada en la prueba-error.

Fase 1: Selección de la información del estado y función de refuerzo

Alternativa 1

La primera aproximación para realizar el aprendizaje por refuerzo la hemos relacionado con los siguientes parámetros:

Estado: Viene definido por los siguientes atributos, la combinación de todos produce 288 posibles estados:

Distancia de Pac-man al fantasma más cercano (computeNearestGhostDistance): Hemos dividido el mapa en regiones en función de la distancia al fantasma más cercano, los valores que definen las distancias son: Cerca, Media y Lejos.

- **Cerca**: Distancia menor de 7 unidades.
- **Media**: Distancia entre 7 y 15 unidades.
- **Lejos**: Distancia más de 15 unidades.

Orientación de Pac-man (computePacManOrientation): La dirección de la que procede Pac-man, que puede ser Norte, Sur, Este u Oeste.

Distancia a la comida más cercana (computeNearestDotDistance): De la misma manera que la distancia al fantasma más cercano, hemos definido los mismos rangos de valores: Cerca, Media, Lejos.

- **Cerca**: Distancia menor de 7 unidades.
- **Media**: Distancia entre 7 y 15 unidades.
- **Lejos**: Distancia más de 15 unidades

Dirección en la que se encuentra el fantasma más cercano (computeNearestGhostDirection): Hemos escogido este atributo para los estados porque Pac-man se comportara de distinta manera dependiendo de donde se encuentre el fantasma objetivo, las direcciones que hemos definido han sido: Norte, Noreste, Este, Sureste, Sur, Suroeste, Oeste, Noroeste. Estos valores los hemos obtenido teniendo en cuenta la posición de Pac-man y del fantasma en el mapa y codificando las posibilidades.

Acciones: Las acciones que puede realizar Pac-man son Norte, Sur, Este u Oeste, en este caso no se permite Stop dado que no aporta nada para alcanzar el objetivo e incluso es peor por que pierde un tick no moviéndose .

Función de transición: Si es legal la acción pedida se realiza, en otro caso, se queda en el mismo estado, actualizando la tabla Q con los nuevos valores de refuerzo y se vuelve a estimar la acción a tomar.

Función de refuerzo: En cuanto a la recompensa que recibe en cada estado nuestro agente hemos desarrollado las siguientes funciones:

- **sumaFantasma**: función que recompensa al agente con la puntuación de comerse un fantasma en ese estado, 200 de refuerzo. Se lanzará cuando el cambio en el score sea de 199 unidades.

- **sumaComida**: función que recompensa al agente con la puntuación de comerse un pac-dot que produce un cambio 99 unidades en la puntuación, si en ese estado se ha comido uno se le recompensa con 100 de refuerzo.

- **sumaAcercarseF**: función que recompensa al Pac-man por acercarse al fantasma más cercano, la recompensa se da cuando se transita entre las distintas regiones con las siguientes recompensas, para obtener las distancias usamos las que nos proporciona el propio juego:
    - **Lejos a medio:** 3.5
    - **Medio a cerca:** 7
    - **Medio a lejos:** -1
    - **Cerca a medio:** -1.5

- **sumaAcercarseC**: función que recompensa al Pac-man por acercarse a la comida más cercana, al igual que con la de fantasmas se recompensa al transitar entre zonas, pero en este caso la recompensas son la mitad.
    - **Lejos a medio:** 1.75
    - **Medio a cerca:** 3.5
    - **Medio a lejos:** -0.5
    - **Cerca a medio:** -0.75

- **sumaIrDireccionCorrecta**: función que premia a Pac-man por ir en dirección hacia el fantasma más cercano, la recompensa por aproximarse es de 1, se detectara según si la dirección en la que mire coincide con alguno de los puntos cardinales hacia el que se encuentre el fantasma.

- **tick**: Por cada tick de juego que pasa se castiga a Pac-man con -1 unidad.

- **penalizaciónDistancia**: Por cada unidad de distancia que está lejos el Pac-man en el estado se le resta 0,1.

- **choquePared**: Si Pac-man se encuentra al lado de un muro que le impide acercarse al fantasma más cercano se le castiga con -300, para evitar que se quede constantemente

chocando con este. Se utiliza la función hasWall para detectar el muro y el atributo dirección del fantasma más cercano para ver cuando se produce esta situación.

Alternativa 2

Nuevos estados: Hemos eliminado la orientación del Pac-man ya que hemos supuesto que generaba bastantes problemas de reconocimiento de estado por nuestra parte, además la función de refuerzo que implicaba este atributo era bastante débil por lo que hemos decidido eliminarlo.

Para suplir la eliminación de este atributo hemos hecho de la distancia al Pac-man un atributo más sensible añadiendo nuevos grados de cercanía. Un problema bastante grande que nos hemos encontrado ha sido en el tercer mapa, ya que el callejón sin salida donde se encuentra unos de los fantasmas es un obstáculo bastante grande a superar ya que alejarse del fantasma más cercano es fatal para nuestro agente.

Para intentar relacionar al agente con los muros hemos creado un nuevo atributo, una lista de 4 booleanos que nos dice si hay algún muro en alguna de sus 4 posiciones.

- **Distancia de Pac-man al fantasma más cercano (computeNearestGhostDistance2):** Se calculará en cuál de estas regiones se encuentra observando cual es la menor distancia a los fantasmas vivos.
    - **Muy cerca**: Distancia menor de 4 unidades.
    - **Cerca**: Distancia entre 4 y 7 unidades.
    - **Medio**: Distancia entre 7 y 10 unidades.
    - **Lejos**: Distancia entre 10 y 13 unidades.
    - **Muy Lejos**: Distancia mayor de 13 unidades.

- **Muros alrededor (nearWallUp, nearWallDown, nearWallLeft y nearWallRight)**: Vector de 4 booleanos para observar si en cada una de las posiciones adyacentes del Pac-man hay muros. Las posiciones a las que pertenecen las direcciones son las siguientes: Siguiendo que cada uno es 1 o 0, 1 si hay muro y 0 si no lo hay, el número que codifica este vector es ‘Arriba Abajo Izquierda Derecha’.

- **Distancia a la comida más cercana (computeNearestDotDistance):** Se mantiene el definido para la primera alternativa.
    - **Cerca**: Distancia menor de 7 unidades.
    - **Media**: Distancia entre 7 y 15 unidades.
    - **Lejos**: Distancia más de 15 unidades

- **Dirección al fantasma cercano reducido (computeNearestGhostDirection2):** En este caso, hemos pasado de 8 posibles direcciones en las que puede identificar la dirección a solo 2, que se identifican si es moviéndose en el eje x o en el eje y. Lo que nos permite que se codifique con un solo bit, el criterio para tomar un valor u otro es:
    - **Eje x:** Si la diferencia entre las dimensiones x del fantasma y Pac-man son menores que las y. Es el valor 0.
    - **Eje y:** Al contrario que el anterior, si las diferencia en el eje y es menor que la del eje x. Tomará el valor 1.

Número de estados: Todas las combinaciones de los atributos de los estados nos dejan un total de 3*2*5*16 = 480 estados.

Función de transición: Si es legal la acción pedida se realiza, en otro caso, se queda en el mismo estado, actualizando la tabla Q con los nuevos valores de refuerzo y se vuelve a estimar la acción a tomar.

Función de refuerzo: La función de refuerzo de este conjunto de atributos se ha basado en la suma de valores de las siguientes funciones:

- **sumaFantasma:** Función que otorga al agente un refuerzo positivo por comerse a un fantasma, el valor con el que recompensa es de 400 unidades.

- **sumaComida:** Función que otorga al agente un refuerzo positivo por comerse un punto de comida, al dar la mitad de los puntos otorga la mitad de refuerzo que comerse un fantasma 200 unidades.

- **sumaAcercarseF2:** Función que recompensa al agente en función de su grado de cercanía al fantasma más cercano, cuanto más se acerca más premia (haciéndolo solo al transitar entre zonas), el castigo de salir de una zona es mayor que el de entrar para que no entre y salga constantemente. Los refuerzos siguen el siguiente esquema:
    - **Muy lejos a lejos:** 1.25 unidades.
    - **Lejos a medio:** 2.5 unidades.
    - **Medio a cerca**: 5 unidades.
    - **Cerca a muy cerca:** 10 unidades.
    - **Muy cerca a cerca:** -20 unidades.
    - **Cerca a medio:** -10 unidades.
    - **Medio a lejos:** -5 unidades.
    - **Lejos a muy lejos**: -2.5 unidades.

- **sumaAcercarseC:** Función que recompensa al agente en función de su grado de cercanía a la comida más cercana, funciona igual que en la alternativa 1.

- **sumaIrDirecciónCorrecta:** Función que refuerza positivamente al agente por ir en la dirección correcta al fantasma más cercano.

- **choquePared2**: Función que penaliza al Pac-man por tener un muro adyacente en la misma dirección que el fantasma más cercano como ocurría en la primera alternativa, pero en este caso hemos buscado que castigue por los muros adyacentes al que el impide ir directo, al igual que el estar acorralado en un pasillo. Para poder hacer lo anterior, mantenemos la detección de muro en la dirección de Pac-man, pero vamos añadiendo castigo según los muros.

  De tal manera que tener un muro en la dirección es un castigo de -40 unidades, -40 más por muros adyacentes y otros -40 si le arrinconan.

- **avoidLoop**: Función que penaliza al agente por volver a visitar un estado que ha visitado en la iteración anterior, evitando bucles, para detectar este evento lo que hacemos es

recurrir al estado siguiente y mirando la dirección que tiene en el estado actual podemos saber de dónde proviene, por lo que sí coinciden es un bucle de longitud 1. Cuando esto ocurre le castigamos con -40 unidades.

- A parte de esto, también se le resta 1 por cada tick debido a la pérdida de score en cada turno.

Fase 2: Construcción del agente

Lo primero que se ha hecho es portar el código necesario del Tutorial 4 que permite que se realice Q-Learning al proyecto de la Práctica 1, para ello se creó una nueva clase en el busterAgents.py con el nombre que se nos indicó QLearningAgent.

En esta nueva clase se introdujeron las funciones necesarias para leer y escribir la qtable de un fichero, calcular la posición de un determinado estado en la tabla Q, inicializar los valores de los parámetros del aprendizaje, cálculo de la función de valor estado Q, el máximo valor de Q de un estado y la que decide qué acción ejecutar.

Aunque solo se muestra una versión de los valores de refuerzo, tanto en la alternativa 1 como en la 2, se han realizado todo tipo de variaciones y ajustes entre los mismos, en las evaluaciones y en la descripción de la fase 1 se han empleado aquellos que han dado mejor resultado.

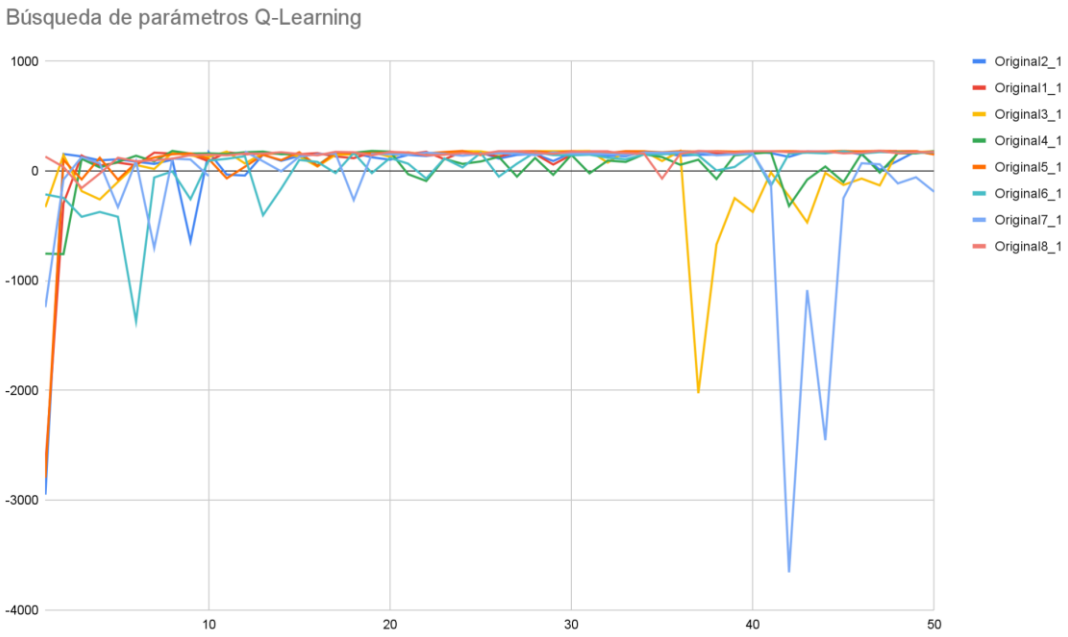
En cuanto a la evolución de los agentes, antes de pasar de la alternativa 1, hemos cambiado numerosas veces los valores de alfa, gamma y épsilon; aparte de esto, la función de refuerzo ha cambiado muchas veces, cambiando valores constantemente y añadiendo funciones que daban poco o ningún valor a la solución. Para que todo esté histórico de cambios no acabe consumiendo gran parte de la memoria, hemos optado por simplemente mencionarlo, ya que la gran razón por la que optamos por cambiar de alternativa fue la poca evolución vista en la alternativa 1 con estos cambios, añadiendo el problema de que no sabíamos cómo afrontar el problema de los muros sin un nuevo set de atributos.

Alternativa 1

Lo primero, tras implementar las funciones necesarias para calcular la posición de un estado en la tabla Q y las recompensas fue determinar los valores del factor de aprendizaje, de descuento y épsilon, partimos de 0,05 en cada una y realizamos pequeñas variaciones sobre el mapa labAA1 ejecutando 50 veces el escenario.

Las pruebas que realizamos fueron las siguientes:

Original1_1: alfa 0,05, gamma 0,05 y épsilon 0,05
Original2_1: alfa 0,05, gamma 0,05 y épsilon 0,10
Original3_1: alfa 0,05, gamma 0,10 y épsilon 0,05
Original4_1: alfa 0,10, gamma 0,05 y épsilon 0,05
Original5_1: alfa 0,05, gamma 0,01 y épsilon 0,05
Original6_1: alfa 0,01, gamma 0,05 y épsilon 0,05
Original7_1: alfa 0,05, gamma 0,05 y épsilon 0,02
Original8_1: alfa 0,05, gamma 0,05 y épsilon 0,07



Como se puede ver inicialmente todas las variaciones son malas, pero la mayoría en cuestión de 3 o 4 escenarios aprende a resolverlo óptimamente y estabilizan las puntuaciones que se obtienen.

Los grandes picos de algunas pruebas son los que corresponden a una subida de la aleatoriedad o de no tener en cuenta la recompensa futura, como era de esperar hay escenarios muy buenos y otros desastrosos en lo que no hace otra cosa que dar vueltas.

El que mejores resultados nos proporcionó con una media de 142 puntos fue el Original_8_1, a partir de este seguimos desarrollando y ajustando los pesos de las recompensas para que funcionase mejor en el primer y segundo mapa, quedando los valores como se han descrito en primera fase.

Los resultados en el segundo mapa fueron prácticamente idénticos al primero, aprende una ruta al fantasma de la derecha y otra al de la izquierda, y con el tiempo hace con más frecuencia la de la derecha.

El problema con el que nos encontramos cuando quisimos pasar a probar el tercer mapa fue que no era capaz de superar el muro que encierra al fantasma superior, aprendía a comerse a los de abajo, pero en cuanto se los comía como el más cercano estaba justo encima no paraba de chocarse con el muro quedándose atascado, a pesar de recibir una recompensa negativa cuando se chocaba con este.

Nos resultó bastante interesante que este modelo de atributos no tiende a confundirse cuando está equidistante a dos fantasmas, simplemente ejecuta una de las acciones posibles y al cambiar los valores de distancia se decanta por uno, esto hace que en mapas con muchos fantasmas esta implementación pueda ser bastante óptima.

Alternativa 2

En cuanto a esta alternativa, se ha mejorado bastante la actuación del agente. Creemos que la principal mejora ha sido en la elección de atributos, ya que nos ha dado una idea más clara de cómo el agente divide los estados en cada episodio. Por otro lado, se ha hecho un ajuste de los valores de refuerzo, aunque atributos como ϵ, α y γ, ya que estos valores estaban bastante bien razonados en el primer set de atributos.

Como se ha mencionado anteriormente, el problema de los muros es algo que nos ha dejado bloqueados durante bastante tiempo, aunque con la ayuda de los métodos de avoidLoop y choquePared, hemos podido sortear en cierta medida el problema. Para ser más exactos, el problema se produce en el mapa 3, ya que el callejón sin salida que hay es bastante largo y alejarse del fantasma más cercano para salir de dicha calle es bastante complicado ya que va en contra de todos los principios del agente. En cambio, para muros de poca longitud es bastante bueno, ya que en el cuarto mapa los llega a resolver de manera factible, ya que no debe tener en cuenta el alejarse en gran medida del fantasma. La mejora ha sido bastante notable en cuanto a los muros, y la resolución de mapas anteriores se ha mantenido intacta.

Aun así, creemos que podría mejorar mucho más el aprendizaje del agente y su manera de superar los muros si se estableciera un tamaño de mapa estándar, de esta manera dentro de los estados se podría indicar la posición de Pac-man en cada momento, lo que permitiría que cada acción que realiza le llevará a un estado distinto y se evitaría el problema de que castigue estados porque cumplen las mismas condiciones que otro.

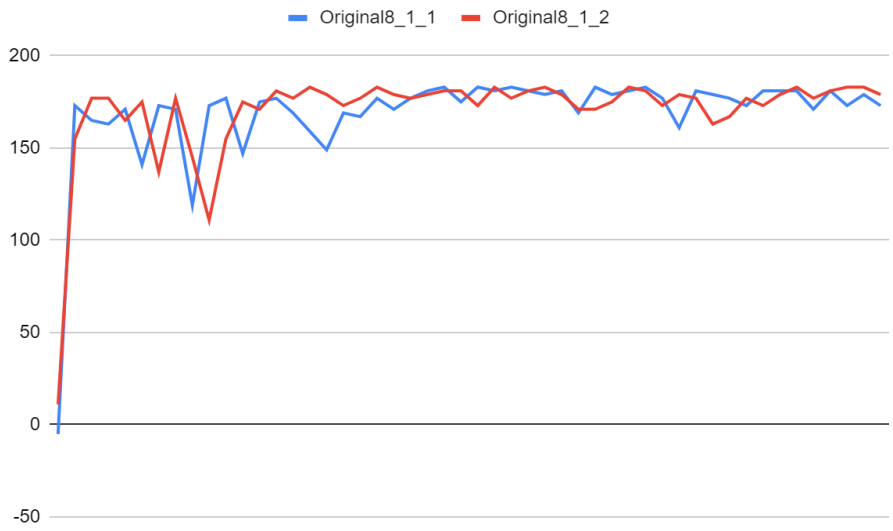
Fase 3: Evaluación de agente

La codificación de las pruebas que se ha seguido sigue el siguiente esquema nom_map_try:

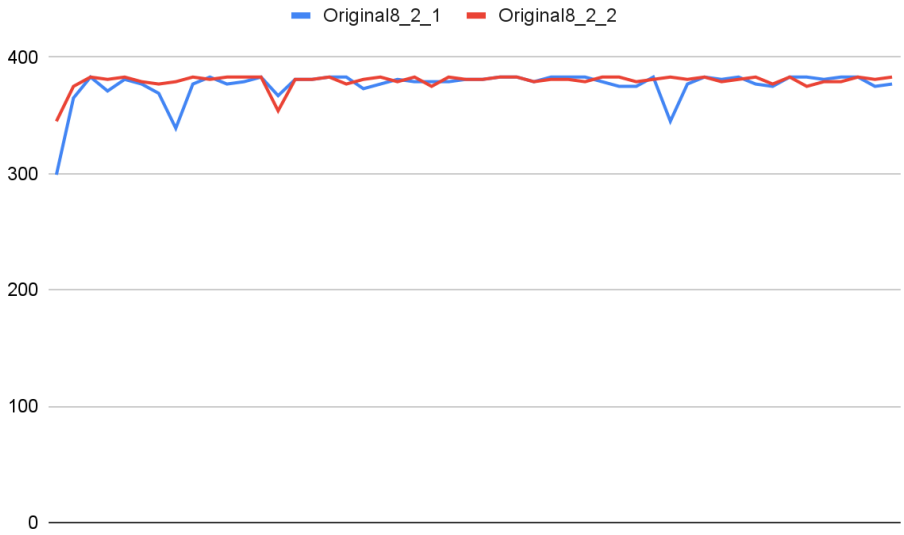
- **Nom**: hace referencia a cuál de las alternativas se trata, la primera se llama Original y la segunda Segundo.
- **Map**: Indica el número de mapa de pruebas realizado. Qué son los siguientes:
  - **1**: Se trata de un escenario abierto en el que solo hay un fantasma, Pac-man y el fantasma se encuentran en esquinas opuestas.
  - **2**: Igual que el anterior, 1, pero en este caso hay otro fantasma extra en la parte superior.
  - **3**: En este mapa se incorpora un muro largo que separa un fantasma en la parte superior de otros dos que se distribuyen en la parte de abajo.
  - **4**: En este mapa se incorporan las comidas además de otros muros para separar fantasmas, cada fantasma se encuentra entre muros para dificultad que acceda directamente a estos.
  - **5**: Es una ampliación del mapa 4 en el que encierra más a los fantasmas y comidas, provocando que se tenga que buscar caminos más enrevesados para lograr ganar.
- **Try**: Indica el intento del que se trata, dado que se han realizado varias ejecuciones sobre el mismo mapa y alternativa.

Alternativa 1

Teniendo en cuenta las ejecuciones del primer mapa, esta alternativa lo resuelve bastante bien, no hay mucho que decir en cuanto a este mapa ya que es bastante simple. Lo que es relevante es ver que esta alternativa intenta buscar en nuevas alternativas de camino óptimo, y acaba resultando en que algunas veces, o bien utiliza el camino de ir arriba y luego a la izquierda, y otras la de derecha, y luego arriba.



El segundo mapa tiene más o menos el mismo comportamiento que el anterior. Es cierto que hay veces que se puede quedar en un bucle cuando está equidistante de ambos fantasmas, pero acaba saliendo de dicho bucle en poco tiempo. Por otro lado, el aprendizaje sigue un ciclo bastante bueno ya que en unos cuantos episodios acaba aprendiendo y consiguiendo una puntuación constante.

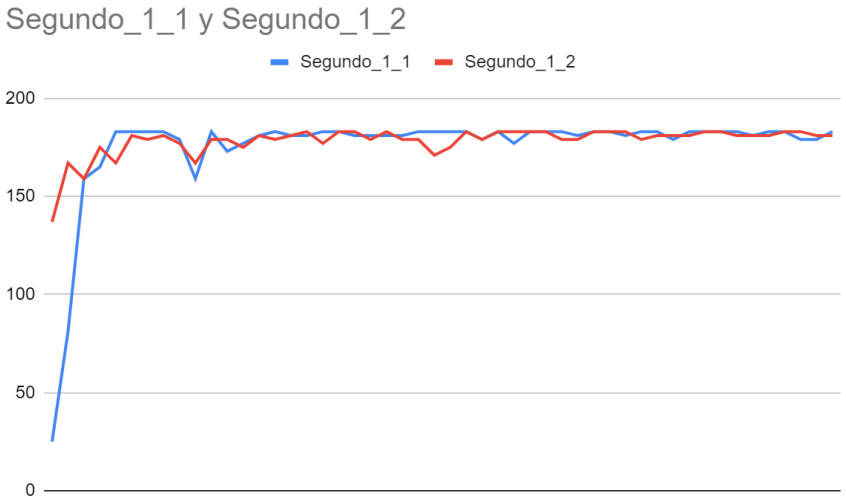


El tercer mapa es el punto donde nos planteamos crear un nuevo set de atributos debido a lo mal que actúa el agente aun habiendo modificado valores y refuerzos. Básicamente el problema radica en el callejón donde se encuentra un fantasma. Llegar hasta ahí desde abajo es bastante complicado ya que debe aumentar su distancia con el fantasma más cercano, y si llega a comerse primero al fantasma del callejón, ocurre lo mismo con los fantasmas que hay fuera de este. Por tanto, llega un punto en la ejecución en el que el Pac-man se queda en un bucle infinito del que no puede salir ni siendo ayudado por la aleatoriedad. Es por esto por lo que decidimos crear la alternativa 2 e intentar solventar el problema de los muros.

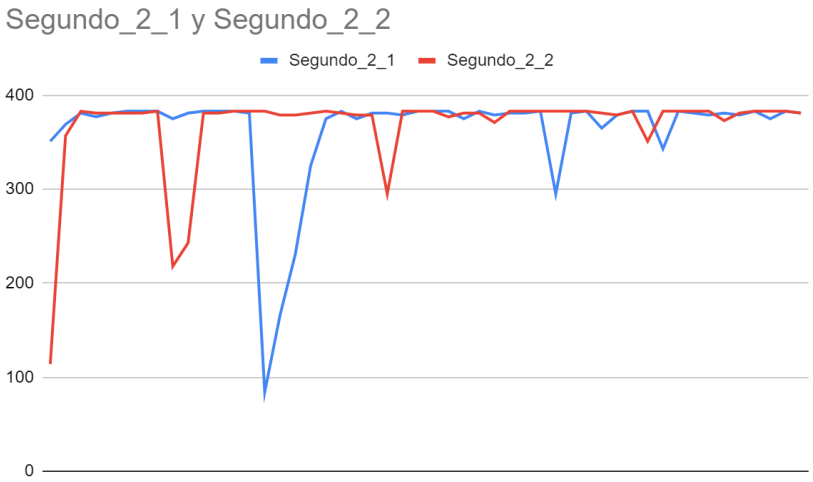
Alternativa 2

A continuación se mostrarán un par de ejecuciones de esta alternativa sobre los mapas que se nos han proporcionado para realizar la evaluación del agente.

Empezaremos con el primer mapa, que como ocurría en la primera alternativa el primer la primera ejecución es la peor, dado que todavía no tiene conocimientos sobre el mapa, pero una vez que lo ha resuelto 1 o 2 veces es capaz de resolverlo muy rápido, con excepción de unos pocos casos en lo que la aleatoriedad le hace tomar rutas alternativas.



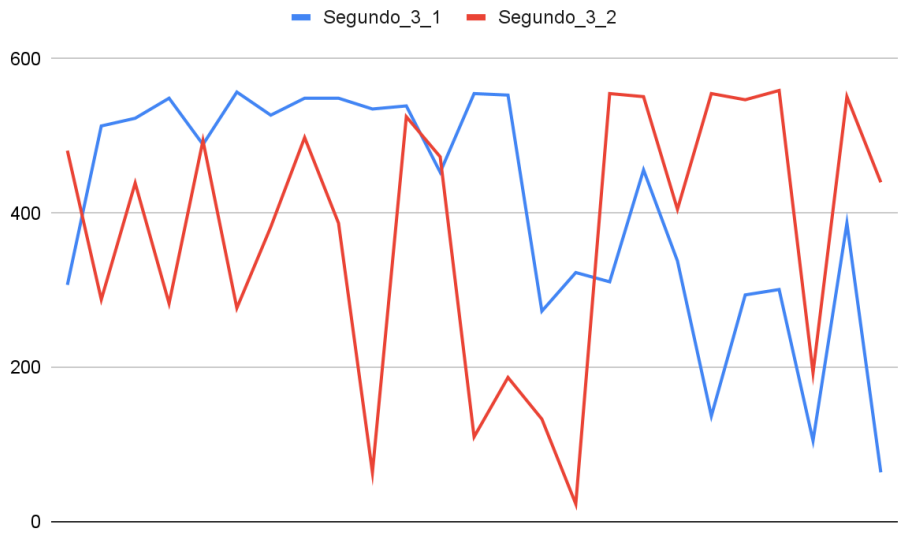
Tras los buenos resultados en el primer mapa se pasó a realizar pruebas sobre el segundo, en el que al igual que en el primero se obtuvieron buenos resultados en general, aunque en este caso en los momentos que se realizaba una acción aleatoria que le hacía acercarse más al otro fantasma provocaron que explorara más el mapa, por lo que a pesar de dar un peor resultado en esa ejecución permitía que fuese más versátil en las situaciones que se le venían.



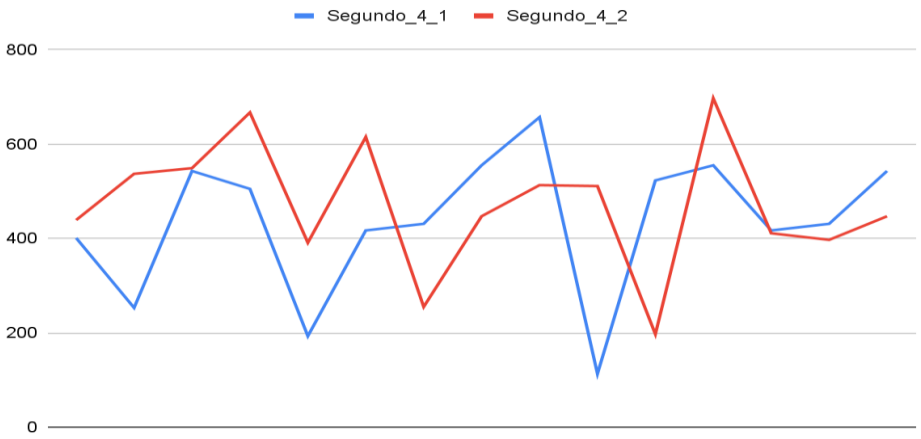
A continuación pasamos a probar nuestro agente en el tercer mapa en el que hay un largo muro en la parte superior que separa dos de los fantasmas de otro que se encuentra acorralado en el extremo del callejón de la parte superior. Como ocurría en la primera alternativa Pac-man le cuesta entender cómo acceder al fantasma que se encuentra al otro lado del muro, aunque en este caso no se queda constantemente en dándose contra el muro y sin salir de ese bucle.

Al soltar a Pac-man en este mapa la primera vez se come rápidamente a los fantasmas de la parte de abajo, que son más accesibles, y trata sin éxito acercarse al otro fantasma a través del muro, pero gracias choquePared2 aprende que ese no es el camino y acaba realizando una vuelta hasta entrar en el callejo y comiéndoselo.

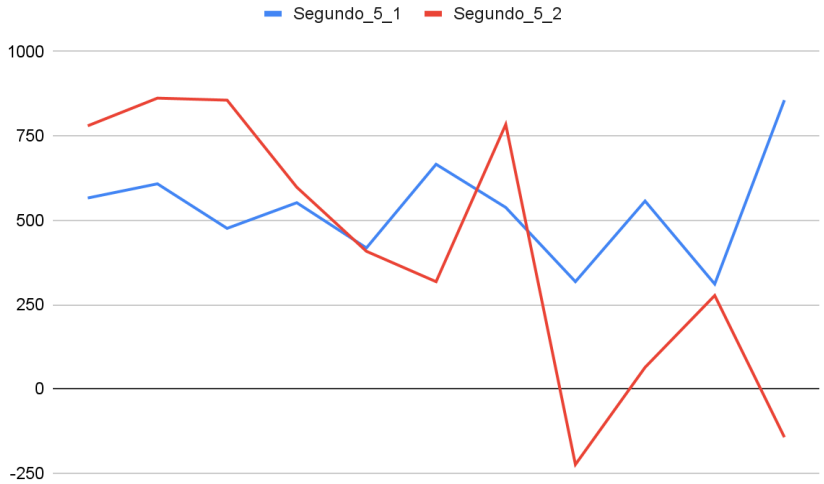
En cuanto al resto de ejecuciones va más o menos aprendiendo como comerse a los fantasmas de abajo y bordear el muro, pero llega un punto en el que la aleatoriedad hace que se pierda en el mapa y le cuesta más superar el muro.



Con el cuarto mapa se nota una mejoría debido a que los muros que hay son bastante más cortos por lo que el agente puede entrar y salir con una mayor facilidad. Un aspecto para tener en cuenta es que una vez Pac-man se come los puntos, el mapa cambia totalmente, generando nuevos estados por lo que debe aprender ‘de cero’ dos veces. Es por esto por lo que el proceso de aprendizaje en este mapa es bastante más largo que en los demás. Aun así, estamos bastante contentos con las ejecuciones realizadas por el agente mostradas en la gráfica.

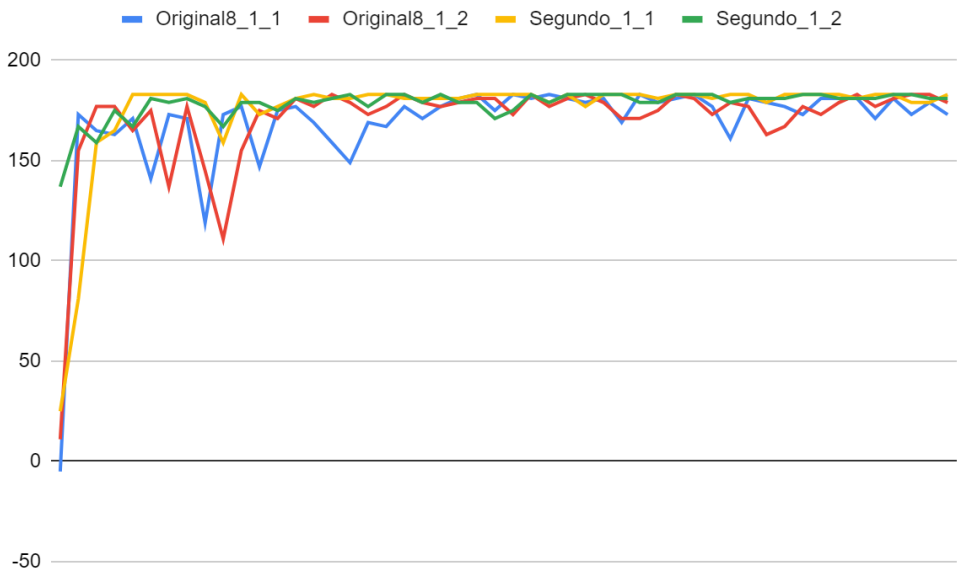


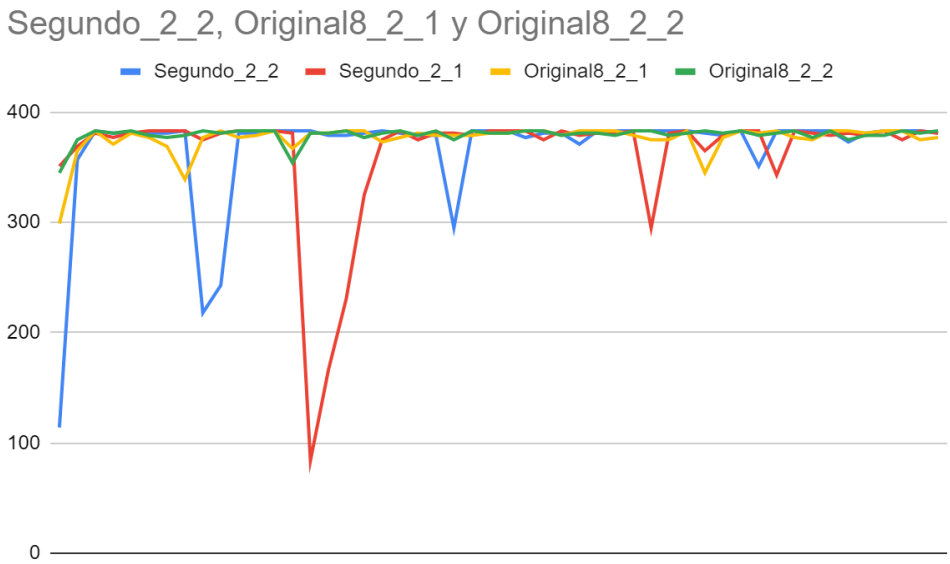
En cuanto al quinto mapa, los resultados no son muy representativos debido a que la estructura del mapa es bastante complicada, la única manera de comer a un fantasma es entrando por un único hueco lo que dificulta mucho que el agente consiga una puntuación aceptable en intentos seguidos. No es de los mapas en los que mejor funciona nuestro agente pero ha conseguido tener algunos episodios bastante aceptables. Aun así, aprender el mapa de manera eficiente es algo que está bastante lejos de los intentos que hemos realizado, que pueden ser vistos en la siguiente gráfica.



Comparación entre Alternativa 1 y Alternativa 2

Tras haber analizado cada una de las alternativas por separado podemos ahora ver cuál de ellas es mejor en cada situación. A continuación unas gráficas comparativas entre ellos:





Como la primera alternativa no es capaz de superar el tercer mapa de manera consistente no se comparará con la segunda que si lo logra, lo mismo para el mapa 4 y 5.
Lo que podemos ver para los dos primeros mapas, es que a pesar de que la primera alternativa no es capaz de superar muros es consistente cuando logra aprender una buena ruta, de más en el segundo mapa es más consistente que la segunda.

En definitiva la segunda alternativa es la mejor porque es capaz de adaptarse a mapas más variados, aunque para los mapas más simples al tener muchos factores en cuenta puede tener episodios más exploratorios buscando rutas alternativas.

Conclusiones

En definitiva, con esta práctica nos hemos dado cuenta de que la elección del espacio de estados es vital para realizar aprendizaje por refuerzo. Por otro lado, la función de refuerzo debe estar equilibrada e intentar suplir las limitaciones que da el agente de búsqueda. Aun así, creemos que hemos conseguido una aproximación bastante satisfactoria respecto a las anteriores, aprendiendo sobre lo que hemos hecho mal e intentando solucionarlo.

A pesar de esto, algunos mapas con muros muy grandes se nos han seguido resistiendo aun habiendo probado diversas combinaciones, de todas maneras, el aprendizaje por refuerzo acaba estando basado en la prueba y el error, y eso es lo que hemos aplicado durante toda esta práctica.