Aprendizaje Automático

Práctica 2: Desarrollo de un agente mediante aprendizaje reforzado

Juan José Blanco Sánchez 100303601

Virginia Hoyas Punzón 100363872

INDICE

Contenido

1. Introducción

En esta práctica se va a llevar a cabo la implementación de un agente de aprendizaje automático mediante aprendizaje reforzado, concretamente, utilizando el algoritmo Q-Learning.

Este tipo de aprendizaje se basa en la obtención de refuerzos positivos o negativos por parte del agente de forma que sea capaz de llevar la tarea propuesta a cabo con eficiencia.

Como se desarrolla más adelante, dicha eficiencia depende de distintas variables a tener en cuenta, tales como los estados por los que puede pasar el agente, que será lo que le lleve converger en cada mapa, las acciones que puede llevar a cabo y las variables *alfa*, la tasa de aprendizaje, *gamma* que representa el descuento en el aprendizaje y *épsilon* que es la probabilidad en cada estado de ejecutar una acción aleatoria de entre las disponibles. El equilibrio entre estas variables será lo que lleve a la tabla Q a converger de forma eficiente.

El objetivo del agente será el de maximizar la puntuación obtenida, es decir, ser capaz de comerse todos los fantasmas en el menor tiempo posible

2. Conjunto de atributos y estados

Los estados vienen representados por una cadena de 7 números, de los cuales los 6 primeros serán 0 o 1 y el séptimo 0, 1 o 2. Cada número representa lo siguiente:

1. Fantasma más cercano al este (1) o al oeste (0)
2. Fantasma más cercano al norte (1) o al sur (0)
3. Pacman tiene una pared al Norte
4. Pacman tiene una pared al Sur
5. Pacman tiene una pared al Este
6. Pacman tiene una pared al Oeste
7. La distancia de Pacman al fantasma más cercano es mayor que 6 (0), entre 2 y 6 (1) o una unidad (2).

Esta definición de los estados dará un total de 192 estados, aunque como se explicará más adelante no se visitarán todos, lo que produce cierta bajada en el rendimiento. Sin embargo, permiten que Pacman termine la mayoría de las partidas.

3. Descripción del código

Mucha parte del código de la práctica es reutilizado del Tutorial 4, puesto que muchas funciones son para coger información de la tabla Q, por lo que no ha dio necesario implementarlas de nuevo.

Las funciones implementadas o modificadas a partir de las originales han sido las siguientes:

computePosition

En esta función se habla de posición como el estado en el que se encuentra el agente, es decir, la posición de la tabla Q.

Para detectar en qué estado se encuentra Pacman, se realiza el siguiente análisis:

1. Buscar el fantasma más cercano.
2. Encontrar la posición del fantasma más cercano respecto a Pacman (N/S y E/W).
3. Comprobar las paredes que se encuentran adyacentes a Pacman (N, S, E y W).
4. Analizar la distancia entre Pacman y el fantasma.

Esta función devolverá una lista de números que representa el estado en el que se encuentra el agente y se comparará con el total de estados generados mediante la función *generateStates* para devolver la posición en la tabla Q en la que se encuentra y acceder a las posibles acciones.

generateStates

La única función de este método es generar todos los estados posibles dentro de los límites de la definición. Mediante fuerza bruta, se enumeran las cadenas de longitud 6 que se pueden generar mediante 0 y 1 y se multiplica escalarmente con las cadenas de longitud 1 que se pueden generar con los números 0, 1 y 2. El total de posibilidades generadas con este método es 192, que es total de estados posibles.

Finalmente, se crea un diccionario en el que la clave será un número entre 0 y 191, que denota el número del estado y que tiene asociado una de las cadenas generadas anteriormente.

Mediante este método, el programa es capaz de leer la cadena del estado y buscar el número que lo representa, pudiendo así leer la posición de la tabla Q.

Update

Esta función será la encargada de actualizar los valores de la tabla Q, es decir, se encargará del aprendizaje del agente, por lo que resulta crucial.

Al contrario que en el Tutorial 4, no habrá estado final, por lo que el la expresión para la actualización de la función será siempre la misma.

getNextState

Esta función sirve como auxiliar para calcular los estados siguientes a los que puede acceder Pacman desde el estado actual, por lo que analizará las acciones que puede tomar en un estado y dependiendo de esta devolverá la poición de la tabla Q para dicho estado y acción.

getReward

Esta función será la utilizada para calcular la recompensa que se adquiere en cada estado. Se basa en las mismas ideas que los estados, de forma que estados similares se corresponda con una forma de recompensa.

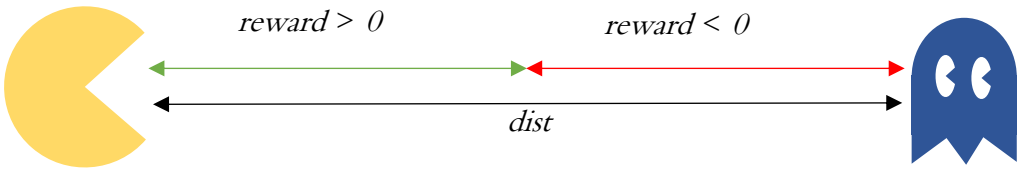
En primer lugar, es necesario hacer cálculos sobre la situación en la que se encuentra el agente, como encontrar el fantasma más cercano o llamar a la función del juego que devuelve si hay pared en una posición.

Es importante destacar el cálculo de la distancia máxima que puede tener pacman de cualquier fantasma, hallando la distancia Manhattan entre el ancho y el alto del mapa. Esto servirá más adelante para conseguir que dependiendo de la distancia, la recompensa vaya variando.

La idea consiste en que parte de la recompensa dada a Pacman dependa de la distancia que hay entre él y el fantasma más cercano. Sin embargo, si solo se analiza la distancia, se obtendrá recompensa negativa tanto cuando está lejos como cuando está cerca, tomado valor 0 cuando se lo come. El planteamiento abordado consiste en desplazar dicho 0 de forma que, llegado a una distancia, empiece a obtener recompensa positiva, fomentando que Pacman se acerque más al fantasma más cercano.

Para esto, se resta a la distancia máxima calculada la distancia entre Pacman y el fantasma y se le suma la distancia mínima en la que se quiere que empiece a dar refuerzo positivo. De este modo, se consigue que, al acercarse más a un fantasma, si está fuera del radio mínimo, la recompensa sea negativa y una vez se acerque lo suficiente, empiece a recibir cada vez más alta positiva. Este valor queda almacenado en una

variable llamada *dist* de forma que se pueda utilizar a la hora de dar las recompensas posteriormente, puesto que será necesario para compensar este desplazamiento



A continuación, se realiza una operación para calcular la diferencia de puntuación entre el turno anterior y el actual, que será la base sobre la que se irá concediendo el refuerzo.

El desarrollo de la obtención del refuerzo dependerá de la posición de Pacman y después de otras variables

```
SI Pacman está encerrado entre dos paredes:
    SI Pacman.y == Fantasma más cercano.y:
        SI distancia(Pacman, Fantasma más cercano) >6:
            reward = reward – dist - 30
        O SI 6 ≥ distancia(Pacman, Fantasma más cercano) ≥ 2:
            reward = reward + dist + 20
        O SI distancia(Pacman, Fantasma más cercano) < 2:
            reward = reward + dist + 25
```

De esta forma se consigue que se reciba una penalización siempre que esté entre dos paredes, evitando verse en esa situación, pero en caso de detectar un fantasma cerca, tanto en distancia como de encontrarse a la mista altura, se compensa dicha penalización puntualmente, de forma que coma el fantasma y salga de estar encerrado entre dichas paredes.

A continuación, se analiza si Pacman está adyacente a alguna pared:

```
SI Pacman está en contacto con alguna pared:
    SI Pacman.y == Fantasma más cercano.y:
        SI distancia(Pacman, Fantasma más cercano) > 6:
            reward = reward – dist - 20
```

```
O SI 6 ≥ distancia(Pacman, Fantasma más cercano) ≥ 2:
        reward = reward + dist + 10
O SI distancia(Pacman, Fantasma más cercano) < 2:
        reward = reward + dist + 15
```

de esta forma, castigando en menor medida al agente cuando está en esta situación, se consigue que busque menos contacto con paredes, pero sin que influya tanto en la recompensa como la peor situación, que sería estar encerrado.

Finalmente, la situación ideal sería en la que no estuviera ni encerrado ni cerca de paredes, de forma que su movilidad sea la más amplia posible, por lo que será la que más se fomente en el agente:

```
SI distancia(Pacman, Fantasma más cercano) > 6:
        reward = reward – dist
O SI 6 ≥ distancia(Pacman, Fantasma más cercano) ≥ 2:
        reward = reward + dist + 25
O SI distancia(Pacman, Fantasma más cercano) < 2:
        reward = reward + dist + 40
```

Por lo tanto, finalmente se tendrá un valor de recompensa que tenga en cuenta las situaciones que se han establecido tanto en los estados como en la propia función de refuerzo y será lo que se devuelva a la hora de llamar a la función.

getAction

Esta será la función llamada desde la clase *Game* y hará que el agente ejecute la acción que le corresponda. Además de inicializaciones de distinto tipo, necesarias para el resto de la ejecución, el mecanismo que permite la ejecución de una acción consiste en, con una probabilidad *épsilon*, ejecutar una acción aleatoria y, en caso contrario, ejecutar la mejor acción para el estado en el que se encuentre Pacman, esto es, el de mayor valor en la tabla q.

4. Agentes desarrollados

Información de todos los fantasmas

En una primera instancia, el agente se compone de información sobre todos los fantasmas, incluyendo sus posiciones y sus distancias. Esto da lugar a una cadena de números de la siguiente forma:

Fantasma 1 al Este (1), Oeste (2) o nada (0) respecto a Pacman.

Fantasma 1 al Norte (1), Sur (2) o nada (0) respecto a Pacman.

Fantasma 2 al Este (1), Oeste (2) o nada (0) respecto a Pacman.

Fantasma 2 al Norte (1), Sur (2) o nada (0) respecto a Pacman.

Fantasma 3 al Este (1), Oeste (2) o nada (0) respecto a Pacman.

Fantasma 3 al Norte (1), Sur (2) o nada (0) respecto a Pacman.

Fantasma 4 al Este (1), Oeste (2) o nada (0) respecto a Pacman.

Fantasma 4 al Norte (1), Sur (2) o nada (0) respecto a Pacman.

El símbolo para nada viene a representar los estados en los que un determinado fantasma ha sido comido y, por tanto, está inaccesible.

Esta representación de estados da lugar a 3^8 = 6561 estados, un número absolutamente inabarcable para recorrer para el agente, por lo que no se implementa esta idea y se pasa a buscar un número mucho más reducido de estados.

Fantasma más cercano

Utilizando esta misma idea, se busca el fantasma más cercano y sus características, de donde se derivan los estados.

La ventaja que implica esta descripción consiste en no necesitar tener en cuenta un estado para el que los fantasmas ya hayan sido comidos, puesto que en el momento en el que Pacman capture un agente este ya no será tenido en cuenta para buscar su posición.

Si analizamos el número de estados obtenidos harán un total de 2^2, de nuevo una cantidad inaceptable de estados por su reducido tamaño que no puede describir los estados por los que pasará el agente, sobre todo en el momento en el que detecte un fantasma detrás de una pared, puesto que querrá dirigirse a este sin tener posibilidad de acceder a él.

Fantasma más cercano y paredes

Para dar una mejor definición de los estados y evitar precisamente los casos en los que el agente se queda bloqueado por ir tras un fantasma detrás de una pared, se añaden cuatro bits, uno por cada dirección posible, para indicar si Pacman se encuentra adyacente a alguna pared. Junto con la función de refuerzo, que dará distintas

recompensas dependiendo de la posición de Pacman con las paredes, hará que sea capaz de distinguir si hay una pared cerca y si va detrás de un fantasma que está por detrás.

Agente final: paredes y distancia

Finalmente, para variar la recompensa obtenida, se añade a la lista un número entre 0 y 2 que defina la distancia entre el fantasma más cercano, como se ha indicado en la sección 2. Esta será la definición de estados que se utilice para la práctica, puesto que el número de estados generados será 192, que resulta un número intermedio que se considera suficiente para la ejecución.

5. Experimentación y resultados

Alpha, Gamma y Épsilon

Alpha representa la tasa de aprendizaje, es decir, cuanto más alta sea con más rapidez aprenderá el agente, pero siendo más sensible a caer en mínimos locales. Por lo tanto, a lo largo de las experimentaciones se ha optado por utilizar valores no demasiado altos, entre 0.4 y 0.6, de forma que, aunque haya que realizar más ejecuciones para que la tabla q converja, se intenta buscar la caída dentro de estos mínimos locales.

Respecto la variable gamma se buscará que el agente tenga más en cuenta las recompensas futuras, de forma que su objetivo sea terminar la ejecución comiéndose los fantasmas lo antes posible en vez de tener en cuenta exclusivamente las recompensas de los estados inmediatos, por lo que se le dará un valor alto entre 0.5 y 0.8

Épsilon resulta la variable más interesante, puesto que será la que produzca la exploración aleatoria de estados. Se corre el riesgo de, en caso de ser muy baja, no explorar suficientes estados por parte del agente y no sea capaz de encontrar políticas óptimas. En caso contrario, puede dar lugar a demasiadas acciones aleatorias, haciendo que el aprendizaje sea más lento y la ejecución infructuosa, puesto que puede encontrarse en un estado cercano a un fantasma y ejecutar una acción que no vaya a aumentar la puntuación comiéndoselo.

Una posible solución a esto es la disminución gradual del valor de épsilon, de forma que al comenzar la ejecución visite muchos estados aleatorios y conforme se avance el valor vaya disminuyendo, escogiendo más a menudo los valores óptimos para cada estado. La parte negativa de esto llega cuando el agente ha tomado decisiones negativas al inicio de la ejecución, provocando que la partida se alargue sin haber aprendido adecuadamente, quedándose bloqueado en una serie de estados.

Para esta práctica se ha tenido en cuenta esta idea, pero debido a esto último se ha decidido no utilizar esta técnica.

Rendimiento en los mapas proporcionados

Para llevar a cabo la experimentación se ha creado un *script* en Bash, de forma que ejecute cada mapa con el agente de Q-Learning e imprima la puntuación obtenida. En caso de que la ejecución de un mapa lleve más de 5 minutos, se interpreta que las acciones tomadas en esa ejecución no han sido las óptimas y se tendrá en cuenta para los cálculos posteriores.

También se ha llevado a cabo una ejecución con fantasmas estáticos y fantasmas con movimiento aleatorio para comprobar el funcionamiento del agente desarrollado en un entorno más realista de juego.

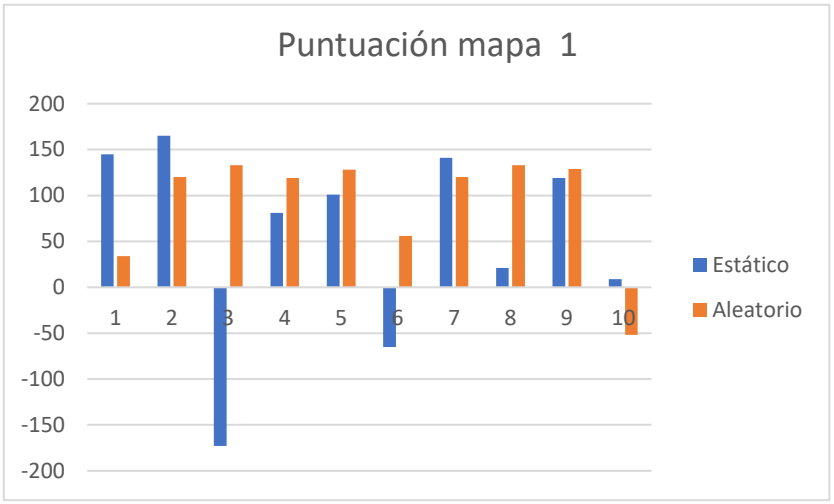
		ronda 1	ronda 2	ronda 3	ronda 4	ronda 5	ronda 6	ronda 7	ronda 8	ronda 9	ronda 10	media	# completados	# completados positivos	% completados positivos
Mapa 1	Estático	145	165	-173	81	101	-65	141	21	119	9	54,4	10	8	80,00
	Aleatorio	34	120	133	119	128	56	120	133	129	-52	92	10	9	90,00
Mapa 2	Estático	325	-190	245	-154	286	-378	349	436	-394	-21	50,4	10	5	50,00
	Aleatorio	254		-183	96	35	351	265	-3	96	153	106,4	9	6	66,67
Mapa 3	Estático	-195	-935	-517				-815	-579			-304,1	5	0	0,00
	Aleatorio	331	258		528	419	168	309	375	468	-508	234,8	9	7	77,78
Mapa 4	Estático	-135	487	-99	59	-813				-97	-137	-73,5	7	0	0,00
	Aleatorio	171	293		65	265	406	-49	392	465	-235	177,3	9	6	66,67
Mapa 5	Estático	-132		-328	504		-352	-116	-154	-506		-108,4	7	0	0,00
	Aleatorio	563	-511	258	566	735	483	-191	134	481	-675	184,3	10	7	70,00

Si se observa la tabla, se puede ver cómo en muchas ocasiones se supera el límite de tiempo establecido, interpretando que el agente no ha logrado su objetivo, pero muchas veces en los mapas más difíciles, a pesar de no obtener una puntuación alta, sí que consigue comerse todos los fantasmas antes de terminar la ejecución. Estas situaciones se dan sobre todo en los casos en los que los fantasmas están estáticos.

Por el contrario, cuando se utilizan fantasmas aleatorios, las situaciones que se dan son más variadas y el cálculo de los estados hace que se den menos bloqueos y se evita encontrar mínimos locales, lo que produce que se acaben las partidas casi siempre e incluso con puntuación positiva.

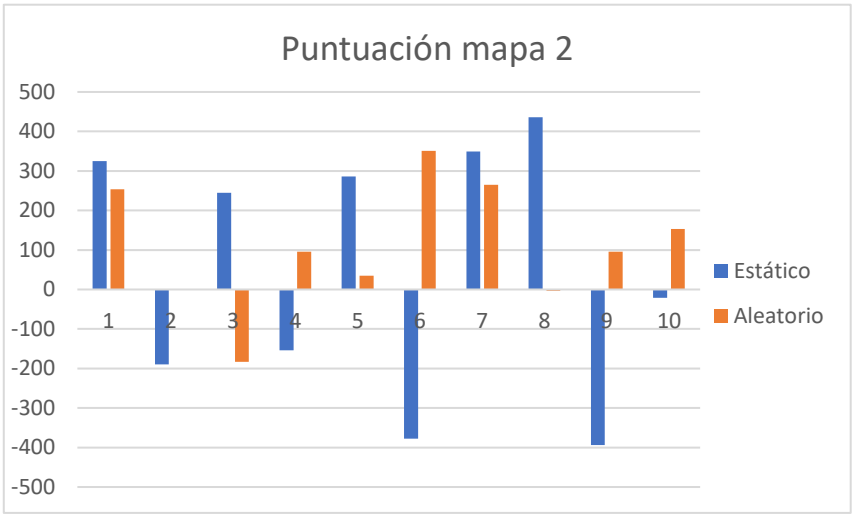
Mapa 1

Este mapa es el más básico, con un único fantasma para ser capturado, por lo que será el primero en ser ejecutado para que Pacman aprenda los estados básicos en los que se puede encontrar.



Mapa 2

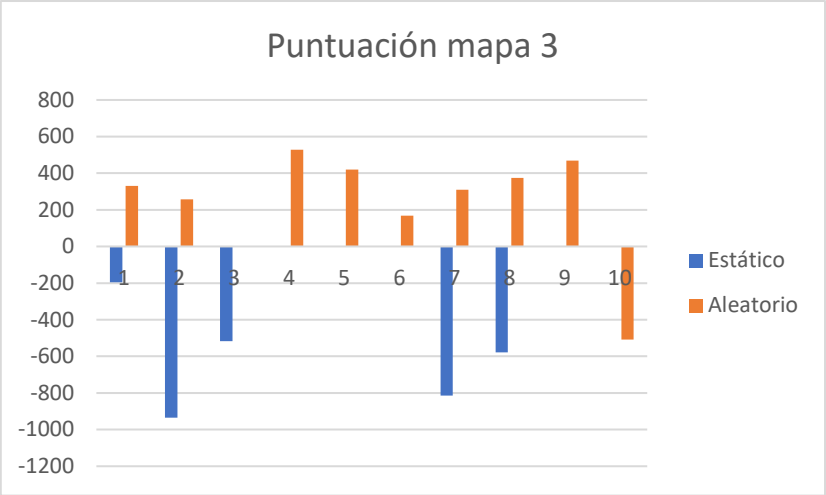
Este mapa es igual que el anterior, pero incluye dos fantasmas, por lo que entra en funcionamiento el mecanismo para detectar el fantasma más cercano.



Mapa 3

Este mapa resulta el más complicado para el agente desarrollado debido a la pared que se encuentra en la parte superior izquierda, pues genera un pasillo muy largo del que es muy difícil escapar.

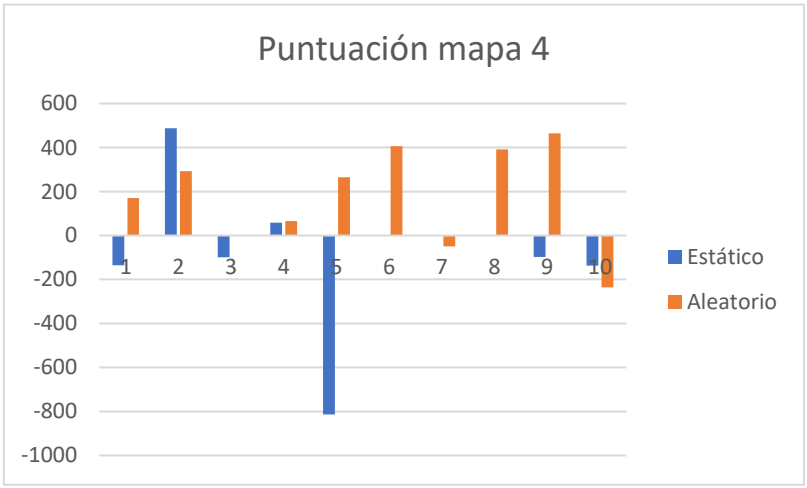
Normalmente, si el agente entra al principio de la partida no podrá volver a salir, pero si lo hace al final, dado que el fantasma está dentro, podrá ganar la partida.



Resulta llamativa la gran diferencia entre los casos en los que los fantasmas son estáticos respecto a los aleatorios, puesto que no solo consigue terminar, sino que además lo hace con puntuación positiva, mostrando una gran diferencia entre ambos casos.

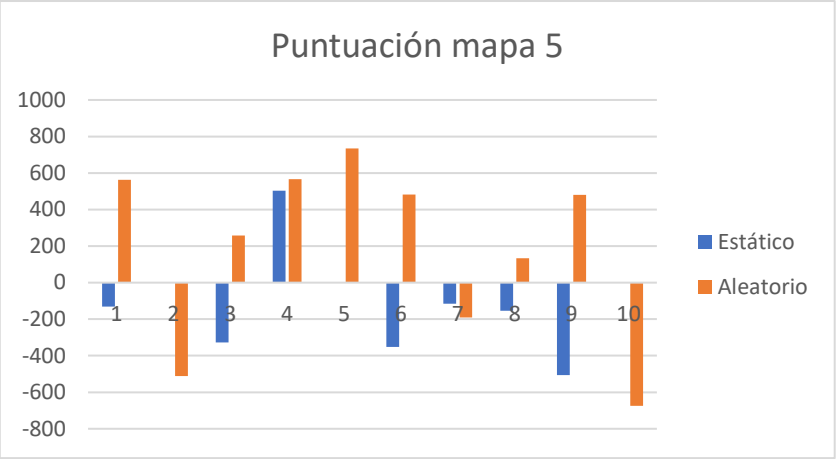
Mapa 4

Este mapa también tiene pasillos, pero puesto que no son tan largos como los del mapa 3, Pacman podrá salir de ellos, aunque con cierta dificultad. Sin embargo, este mapa puede ser completado por el agente más veces que el tercer mapa.



Mapa 5

En este mapa, además de pasillos, también se encuentran huecos en los que Pacman corre el riego de quedar atrapado, pero como el mapa 4, dado su tamaño es posible para Pacman poder salir de ellos y seguir ejecutándose.



Este mapa resulta el más complejo y se demuestra en esta gráfica, observándose su disparidad de resultados, en la que se puede acabar tanto con puntuación positiva como negativa o incluso no acabándola.

6. Conclusiones

Mejoras

Update

Se sospecha que se puede afinar más la función de actualización diferenciando entre estados más deseables que otros, de forma que al entrar en una serie de estados la función de actualización genere mejores o peores valores para la tabla q a conveniencia.

Estados

Si se observa la tabla q, se puede observar que hay muchos estados por los que no se pasa, por lo que es de esperar que haciendo una poda de estados se pueda reducir la tabla q y se haga el desarrollo de manera más eficiente, fomentando la convergencia, pudiéndose incluir otras definiciones a los estados, equilibrando por tanto el número de estados con sus definiciones.

Refuerzo

A pesar del esfuerzo invertido a la hora de definir la función de refuerzo, cabe esperar que de dar refuerzos aún más exhaustivos se consiga un mejor rendimiento del agente.

Bolas

Para el desarrollo de este agente no se han tenido en cuenta las bolas, puesto que el objetivo es terminar la partida lo antes posible haciendo que Pacman capture a todos los fantasmas lo antes posible.

Sin embargo, en mapas como el 5 esto puede provocar que el fantasma presente una incoherencia, puesto que en una posición en la que no hay nada para él, porque no detecta las bolas, recibe un refuerzo desmesurado. Aunque no se ha sabido solventar esta situación, resulta una cuestión a tener en cuenta para evitar dicho desfase de puntuación.

Apreciaciones generales

Esta práctica nos ha puesto en contacto con la idea del aprendizaje por refuerzo, concretamente en el entorno de un juego como es una versión de Pacman en la que el objetivo es comer todos los fantasmas, en lugar de ser como la versión original, en la que se debe comer todas las bolas y evitar ser capturado por los fantasmas, a no ser que coma una de las bolas más grandes, en cuyo caso podrá comérselos y retrasar así su futura captura mientras sigue comiendo bolas pequeñas.

Esta idea nos acerca al uso de este tipo de aprendizaje en otro tipo de videojuegos, concepto que ya se ha utilizado en situaciones tales como desarrollar un bot que juegue a Starcraft 2 (AlphaStar) o DOTA 2 (OpenAI), mediante el uso de la variante profunda conocida como Deep reinforcement learning. Y no solo en videojuegos, sino en juegos de mesa tradicionales como el ajedrez o el Go, hecho reconocido mundialmente tras el desarrollo y el éxito de AlphaGo.

Esto da una idea del potencial del aprendizaje por refuerzo a la hora de ser implementado en entornos con un alto grado de variabilidad y complejidad, sobre todo debido a su interacción tan directa con seres humanos altamente especializados en una tarea.

Cabe esperar que se pueda ejecutar un agente de este tipo en entornos reales mediante sistemas de asistentes virtuales o robots que tengan que llevar a cabo una tera concreta. Sin embargo, como se ha comprobado en esta práctica, hay que ser muy cuidadoso con los estados y las interacciones que pueda ejecutar el agente para evitar problemas que desencadenen resultados no deseados.

Opiniones sobre la práctica

En esta práctica nos hemos familiarizado con el aprendizaje por refuerzo y el algoritmo de Q-Learning. A pesar de tener que desarrollarlo para realizar una tarea sencilla, el tener que apelar a las distintas funcionalidades del programa resulta un poco farragoso, lo que da lugar a un código un poco confuso.

Sin embargo, una vez hemos encontrado dichas funciones y hemos sido capaces de aplicarlas adecuadamente, el desarrollo no ha sido complicado y hemos podido crear un

agente que, si bien no consigue siempre la puntuación más alta, sí que consigue terminar la ejecución satisfactoriamente.

Además, como se ha indicado en la sección anterior, el aprendizaje por refuerzo tiene usos muy interesantes y curiosos, por lo que la sensación de estar trabajando en un entorno de este tipo resulta muy gratificante.