Aprendizaje automático

Práctica 2

Grupo 84

Sabrina Riesgo Reyes	100363834	100363834@alumnos.uc3m.es
Laura Yunta García	100363785	100363785@alumnos.uc3m.es

Índice

1. Introducción

El desarrollo de esta práctica consiste en la aplicación del algoritmo Q-learning en el dominio del Pac-Man para conseguir que funcione de forma automática en la mayor variedad de laberintos posibles. Por tanto, el objetivo del agente es maximizar la puntuación obtenida.

Finalmente, este documento se dividirá en los siguientes apartados:

- **Conjunto de atributos:** En este apartado se denota la justificación del conjunto de atributos elegido y el rango utilizado para la definición de los estados.
- **Función de refuerzo:** En este apartado se describe la función de refuerzo final empleada.
- **Código generado:** En este apartado se describe el código generado para llevar a cabo el aprendizaje de la política de comportamiento.
- **Agente final:** En este apartado se describe el agente final implementado, indicando la evolución histórica de los agentes implementados hasta obtener el agente final.
- **Análisis de los resultados:** En este apartado se describen y analizan los resultados producidos por el agente final implementado tras la evaluación.
- **Conclusiones:** En este apartado se denotarán las conclusiones, apreciaciones generales, descripción de problemas encontrados y comentarios personales encontrados durante el desarrollo de la práctica.

2. Conjunto de atributos

En este apartado se va a detallar el procedimiento seguido en el momento de escoger los atributos con los cuales se representa el estado del juego, de forma que garanticen que tras la ejecución del juego el pacman haya obtenido la máxima puntuación posible.

Para comenzar, se han tenido en cuenta los atributos iniciales de la práctica anterior, y con el objetivo de conseguir los mejores, se analizaron cada uno de ellos por separado y por último en conjunto. En primer lugar, se descartaron atributos relacionados con posiciones, tanto del pacman como de fantasmas, y con distancias, ya que se ha considerado que son irrelevantes. Además, se han descartado los atributos que indican si junto al norte, sur, este u oeste de pacman hay una pared. Finalmente, se ha representado el estado como una lista con 5 atributos que se detallarán a continuación.

Estado: $[x_1, x_2, x_3, x_4, x_5]$

El atributo x_1 representa la posición del fantasma más cercano con respecto a la posición del pacman. Los posibles valores que puede tomar este atributo son:

- 0: el fantasma más cercano se encuentra al norte del pacman.
- 1: el fantasma más cercano se encuentra al sur del pacman.
- 2: el fantasma más cercano se encuentra al este del pacman.
- 3: el fantasma más cercano se encuentra al oeste del pacman.
- 4: el fantasma más cercano se encuentra al noroeste del pacman.
- 5: el fantasma más cercano se encuentra al noreste del pacman.
- 6: el fantasma más cercano se encuentra al suroeste del pacman.
- 7: el fantasma más cercano se encuentra al sureste del pacman.

Los atributos x_2, x_3, x_4, x_5 representan las acciones legales en función de la posición en la que se encuentra el pacman (norte, sur, este, oeste). Pueden tomar el valor 0, lo que significa que dicha acción no es legal, o el valor 1, lo que significa que dicha acción sí es legal.

Cabe destacar que aunque se ha tenido más en cuenta aquellos atributos que podrían ser relevantes a la hora de formar el conjunto, también se ha intentado reducir el número de filas de la tabla Q lo máximo posible con el fin de evitar aumentar la complejidad computacional del proyecto.

3. Función de refuerzo

Se ha tenido en cuenta como refuerzo la diferencia de puntuaciones entre dos estados. Al llamar al método update desde game.py, es necesario pasarle como parámetro el refuerzo, por lo que se calcula la diferencia entre el estado actual y el siguiente. En caso de ser un valor negativo dicha diferencia, no se penaliza con un valor negativo, simplemente se pasa un valor 0. Además, se ha creado un método llamado getReward que se encarga de premiar la cercanía a los fantasmas. Esta función es llamada desde el método update, en el cual se suma dicho refuerzo al pasado como parámetro desde game.py.

4. Código generado

Se ha creado una nueva clase QLearningAgent como se indica en el enunciado, que contiene todos los métodos necesarios para llevar a cabo el aprendizaje de la política de comportamiento. Los métodos pertenecientes a esta clase son muy similares a los utilizados en el tutorial 4, por lo que a continuación se explicarán los cambios en aquellos que se han modificado y las nuevas funciones creadas.

1. computePosition(self, state): devuelve la fila de la tabla Q correspondiente a un estado determinado. Para ello se transforma el estado que recibe el método como parámetro (state) en un nuevo estado que se obtiene a través de la función getData la cual se explicará más adelante. Finalmente calcula la fila teniendo en cuenta el número de atributos utilizados en el nuevo estado y los posibles valores que puede tomar.
2. update(self, state, action, nextState, reward): se encarga de actualizar los datos en la tabla Q. Se ha añadido un aumento del refuerzo llamando al método getReward que se explicará más adelante y, además, se ha transformado la letra inicial de cada acción ya que en algunos casos empiezan por mayúscula y en otros por minúscula. En el caso de la acción “stop” se ha decido cambiar la acción de forma aleatoria.
3. getData(self, state): devuelve un nuevo estado que contiene el conjunto de atributos del apartado 2 del estado introducido por parámetro. Dichos atributos se obtienen de igual forma que en las prácticas anteriores.
4. getReward(self, state): devuelve un valor de refuerzo para premiar la cercanía del pacman a los fantasmas.

Además, se han realizado cambios en game.py para llamar a la función update. Simplemente se almacenan los valores que posteriormente se le pasarán como parámetros.

5. Agente final

Como se ha ido explicando en apartado anteriores los pasos seguidos para la realización del agente han sido, en primer lugar la elección de un conjunto de atributos. Una vez seleccionados los atributos es necesario realizar una función que se encargue de crear ese nuevo estado. Seguidamente, se debe ir ejecutando repetidas veces cada uno de los mapas por separado. En nuestro caso hemos empezado con el primer mapa y tras varias ejecuciones pasamos a comprobar que funciona correctamente. A continuación se realiza el mismo procedimiento con el segundo mapa, pero esta vez además de comprobar el correcto funcionamiento del agente en este, se comprueba también que sigue funcionando en el mapa anterior, y así sucesivamente hasta abarcar todos los mapas.

Las principales diferencias observadas tras la ejecución de varios conjuntos de atributos nos llevaron a la selección de los atributos finales descritos anteriormente. Inicialmente teníamos como atributos el número de fantasmas vivos y si había paredes alrededor del pacman. Dichos atributos no resultaron ser eficientes ya que el pacman no realizaba las acciones adecuadas y algunas veces entraba en bucle, por lo que decidimos que el número de fantasmas no era relevante y eliminamos dicho atributo. Posteriormente, agregamos como atributos las distancias a los fantasmas, discretizando los valores. Tras calcular la distancia entre cada fantasma y el pacman clasificábamos dicho valor como cercano (distancia menos que 4) o lejano (distancia mayor que 4). Sin embargo, como se puede observar en el conjunto de atributos, estas distancias no aparecen debido a que “cercano” o “lejano” no dice mucho y es un valor relativo ya que al utilizar un mapa muy grande, estar a una distancia de 8 podría ser bueno mientras que en nuestro caso dábamos por hecho que se encontraba lejos. Finalmente decidimos eliminar también las distancias y utilizar un solo atributo que nos dijera si el fantasma se encuentra al norte, sur, este, oeste, noroeste, noreste, suroeste o sureste con respecto al pacman; y con el objetivo de evitar las paredes en el caso de los mapas 3, 4 y 5, introducimos los atributos que nos informan sobre las acciones legales del pacman dada su posición.

6. Análisis de los resultados

En este apartado se va a analizar los resultados obtenidos además de comparar el agente final con el agente desarrollado en el tutorial 1. Para ello se tendrán en cuenta los siguientes datos: número de fantasmas comidos, número de puntos de comida conseguidos, puntuación final y número total de ticks en los distintos laberintos propuestos.

A continuación se muestran los resultados obtenidos con el agente del tutorial 1.

Laberinto	Ticks	Fantasmas	Puntos de comida	Puntuación
labAA1	16	1/1	0/0	183
labAA2	15	2/2	0/0	383
labAA3	28	3/3	0/0	569
labAA4	27	3/3	0/2	571
labAA5	38	3/3	0/7	562

Analizando estos datos se puede concluir que se trata de un agente guiado por los fantasmas, es decir, su objetivo es conseguir cada fantasma en el menor número de movimientos posibles. Sin embargo, no consigue ningún punto de comida, salvo que se encuentre de camino a un fantasma, caso que no ocurre en ninguno de los mapas anteriores. Además, es importante tener en cuenta que en ningún momento se realiza la acción “stop”, por lo que la puntuación no se ve afectada por la realización de acciones innecesarias.

A continuación se muestran los resultados obtenidos con el agente final.

Laberinto	Ticks	Fantasmas	Puntos de comida	Puntuación
labAA1	16	1/1	0/0	183
labAA2	18	2/2	0/0	381
labAA3	-	-	-	-
labAA4	-	-	-	-
labAA5	-	-	-	-

Teniendo en cuenta los resultados obtenidos tras la ejecución del agente final en los cinco laberintos se puede observar que tanto en el tutorial 1 como en el desarrollo del agente final se han tenido en cuenta los fantasmas ya que siempre el número de fantasmas comido es el máximo, además es la condición de fin del juego. Además, se puede observar que la puntuación y el número de ticks es muy similar en ambas pruebas. Esto se debe a que elige el camino más corto posible para llegar al fantasma; cabe destacar que no se ha eliminado la acción “stop” en el agente final, por lo que en algunos casos, como ha ocurrido en la ejecución del segundo laberinto, hay un mayor número de ticks y la puntuación es menor. Finalmente, los tres últimos mapas no funcionan salvo el tercero que funciona pero realizando demasiadas acciones.

7. Conclusiones

Una vez desarrollada la práctica nos ha quedado más claro cómo funciona el aprendizaje por refuerzo ya que el tutorial 4 no trataba el tema con la suficiente profundidad. A continuación, explicaremos los problemas encontrados que no nos han permitido conseguir el correcto funcionamiento del agente final en todos los mapas dados, y los detalles a mejorar sobre lo realizado que permitirían conseguir un agente más eficiente.

- En cuento a las mejoras del agente desarrollado consideramos que, en los mapas sencillos funciona correctamente ya que va a por los fantasmas siguiendo el camino más corto. Sin embargo, eliminar la acción “stop” de la lista de posibles acciones a realizar por el agente, permitiría que en nuestro caso fuese igual de eficiente que el agente del tutorial 1 ya que al quedarse quieto no aporta nada y solo provoca pérdida de puntuación.
- En cuanto al desarrollo de un agente que funcione correctamente en todos los mapas, nos han surgido problemas durante el proceso de aprendizaje. Como se ha explicado en apartado anteriores, se ha comenzado con el primer mapa y al observar que funcionaba correctamente eliminando la exploración aleatoria de acciones, comenzábamos a realizar el mismo proceso de aprendizaje con el siguiente mapa. Al empezar a utilizar el tercer mapa, observamos que poco a poco iba mejorando, por lo que decidimos poner a 0 los valores de epsilon y alfa para comprobar que funcionaba correctamente. Sin embargo, el aprendizaje en el tercer mapa hacía que el correcto funcionamiento del agente en los mapas anteriores cambiara, llegando a entrar en bucle en varias ocasiones. Para corregir este fallo intentamos cambiar nuevamente los atributos, realizar menos ejecuciones en el proceso de aprendizaje y variar los valores de épsilon y alfa, pero no fue posible, nos ocurría nuevamente lo mismo. A pesar de no conseguir que el agente funcionara en los últimos mapas, consideramos que dichos errores podrían ser causados el número de ejecuciones en el proceso de aprendizaje y la similitud entre algunos estados en los distintos mapas.