Aprendizaje Automático

Competición:

Pac-Man con Refuerzo

JOSE MANUEL FRIAS SALVACHUA 100383533

CESAR MANUEL ARROJO LARA 100383510

uc3m | Universidad Carlos III de Madrid

Índice

1. Introducción

En este documento se comentará en términos generales el proceso de creación del agente por refuerzo, guiado por tabla Q. El estado elegido y el algoritmo que permite extraer el estado del gameState.

2. Proceso de creación del agente

Para llegar al agente final hemos pasado por varias iteraciones.

En la primera iteración, el estado tenía dos atributos:

1. **Dirección del fantasma más próximo** con cuatro valores Norte, Sur, Este, Oeste
2. **Si un fantasma estaba cerca** con cuatro valores booleanos para cuatro fantasmas

La distancia se calculaba mediante Manhattan por lo que la dirección era la recta que unía el fantasma más cercano y el pacman, ignorando muros. Para comprobar si un fantasma estaba cerca simplemente mirábamos si su distancia era mayor que 3 o no. La función de refuerzo era la diferencia entre el scoreActual y el scoreSiguiente.

El agente solucionaba mapas simples, pero no podía rodear muros ni buscar pac-dots.

El problema principal eran los muros para ello recurrimos a Distancer, uno de los archivos base del pacman que se encarga de calcular las distancias. Emplea una función basada en el algoritmo de Floyd-Warshall que permite saber la distancia real de un punto del layout a otro teniendo en cuenta las paredes y el camino que habría que recorrer. Por tanto, decidimos hacer el cambio y mirar la dirección desde un punto de vista diferente: qué dirección sería la que nos acercase más al objetivo. La función de refuerzo se mantuvo igual.

En esta segunda iteración acabamos obteniendo un agente que era capaz de esquivar los muros e incapaz de buscar comida.

Este tercer agente añade un nuevo atributo a los dos anteriores:

1. **Dirección del fantasma más próximo** con cuatro valores Norte, Sur, Este, Oeste
2. **Si un fantasma estaba cerca** con cuatro valores booleanos para cuatro fantasmas
3. **Dirección de la comida más cercana** con cinco valores Norte, Sur, Este, Oeste y None

Se le añade además un valor extra para el caso de no quedar comida (por que si no hay comida, la comida más cercana no está en ninguna de las cuatro direcciones). También se cambia la función de refuerzo. Introducimos el refuerzo basado en la distancia que separa el pacman del fantasma o la comida más cercana y cogíamos la menor de las 2.

3. Definición de estados

En la iteración final disponemos de un agente que presenta sólo 2 atributos con 4 y 5 valores respectivamente. Estos son:

- **Dirección del Fantasma más cercano**: Con las posibilidades de Norte, Sur, Este y Oeste. Refina la idea básica que teníamos sobre cómo indicar los fantasmas. En vez de mostrar todos, nos centramos en el fantasma más cercano y cogemos la dirección en la que nos tenemos que mover para acercarnos a él.
- **Dirección de la Comida más cercana**: Con las posibilidades Norte, Sur, Este, Oeste y None. Sigue la misma idea de antes: centrarnos sólo en la comida más cercana y ver en qué dirección nos tendríamos que mover para acercarnos a ella. También tuvimos que mantener la posibilidad de None en este caso ya que puede que haya mapas sin comida o en las que el pacman se lo haya comido todo

4. Obtención del estado

La obtención de datos se realiza fácilmente a través del gameState, transformándolos en una tupla State como las que hemos definido en el punto 3. Para ello:

1. Guardamos el score en la posición 1 del array de [scoreAnterior, scoreActual] utilizando el método gameState.**getScore()**.
2. Invocamos al método **calculateDir()**, que averigua la dirección que acorta la distancia del pacman con el fantasma más cercano. (estado[0])
3. Invocamos al método **foodDirect()**, que averigua la dirección que acorta la distancia del pacman con la comida más cercana o la marca a None si no quedan comidas. (estado[1])

Para calcular la dirección óptima de acercamiento al fantasma más cercano mediante **calculateDir()**:

1. Inicializamos el array “dinamic” a 9999, la representación de distancia máxima. Y llamamos al método **findMinDistance()** para cada una de las 4 direcciones sólo si es legal moverse en esa dirección. Éste último es el método que utilizamos en la práctica 1, simula la nueva posición del pacman si se hubiera movido en esa dirección y, mediante el Distancer, calcula la distancia real que separa la posición calculada del pacman y la del fantasma más cercano.
2. Comparamos las distancias para cada posible movimiento y nos quedamos con la que sea menor ya que el índice de esta representará la dirección en la que ir.

Para calcular la dirección óptima de acercamiento a la comida más cercana mediante **foodDirect()**:

1. Comprobamos que haya comida mediante el método gameState.**getNearestFood**()
    a. Si no hay, termina dejando el valor por defecto 4 cuyo significado es None.
2. En caso de que sí haya comida, llamamos al método **huntFood()** para cada una de las 4 direcciones sólo si es legal moverse en esa dirección. Éste último es una copia de **findMinDistance()**, que opera con la posición calculada del pacman y la de la comida más cercana. Actuaría como una función simplificada de la captura de Fantasmas sólo que en vez de ir guardando las distancias para hacer el cálculo final, actuamos en cada momento de la función.

Cabe destacar que en todo momento mantenemos el estado anterior, pues, para poder realizar el aprendizaje necesitamos un estado, una acción, el estado que se produce como resultado de la aplicación de esa acción y el refuerzo obtenido por la decisión.

5.¿Por qué este agente?

La respuesta es bastante sencilla. Hemos decidido usar el agente obtenido en aprendizaje por Refuerzo ya que nuestro intento de agente usando aprendizaje supervisado no fue fructífero y acababa bloqueándose en loops infinitos. Este agente consigue terminar la mayoría de mapas con resultados bastante aceptables por lo que su elección era obvia.

6.Ejecución

Se ha adaptado el código enviado en la Práctica 2 para que el agente esté contenido en la clase BasicAgentAA por lo que introduciendo de comando “python busters.py -p BasicAgentAA” funcionaría.