Universidad Carlos III de Madrid

Aprendizaje Automático

Computer Science Engineering

Tutorial 1: Plataforma PacMan

Authors:
Daniel Medina García
Alejandro Rodríguez Salamanca

February 10, 2016

Contents

Pregunta 1

¿Qué información se muestra en la interfaz? ¿Y en la terminal? ¿Cuál es la posición que ocupa PacMan inicialmente?

En la interfaz se muestra el tablero con PacMan. En la parte inferior podemos observar los indicadores de qué fantasmas han sido comidos, así como la puntuación de la partida y cuatro números, cada uno del color de un fantasma, que se corresponden con la distancia que hay desde PacMan a los fantasmas

En la terminal no se muestra ninguna información.

La posición que ocupa PacMan al inicio de la partida depende del mapa usado. En ciertos mapas, PacMan se encuentra en la esquina superior izquierda, en otros, PacMan sale en la fila inferior, centrado.

Pregunta 2

Según tu opinión, ¿qué datos podrían ser útiles para decidir lo que tiene que hacer PacMan en cada momento?

Para tener una información completa sobre el estado del juego, lo ideal sería poder tener acceso a las posiciones exactas de todos los agentes (fantasmas y PacMan) así como los obstáculos que pueden hacer variar la ruta hasta cualquier posible objetivo (mapa). Como esto no es posible por restricciones del enunciado, los datos "legales" que podrían interesarnos para decidir qué dirección debe tomar PacMan son:

- **Posición de PacMan**: la posición en la que se encuentra nuestro agente restringirá los movimientos legales que podrá tomar. Así, este parámetro es básico para la implementación.
- **Dirección de PacMan**: la dirección hacia la cual "mira" PacMan nos es muy útil para la comparación que se comenta en el siguiente punto, pues nos indica la última acción realizada por el agente.
- **Distancia de PacMan a los fantasmas**: comparando la distancia hasta el fantasma más cercano antes y después de tomar una acción se puede evaluar si dicha acción fue o no acertada.
- **Contenido de las celda adyacentes a PacMan**: la legalidad de los posibles movimientos depende de si la celda objetivo es o no un muro o un límite del tablero. Si no tiene información sobre las celdas contiguas a sí, el agente podría cometer una falta que cerraría el juego, causando una derrota.
- **Densidad de fantasmas en una zona**: Con este parámetro, podríamos saber si en cierta zona del mapa se encuentran más fantasmas, y tomar la decisión de dirigirnos hacia esa zona en vez de hacia otra zona en la que solo se encuentre un fantasma, pese a que éste se encuentre a una distancia menor, ya que existiría la posibilidad de eliminar más fantasmas. No hemos observado dicho dato en el código del juego, pero es posible que exista la forma de implementar la forma de conseguirlo.

Pregunta 3

Revisa la carpeta layouts. ¿Cómo están definidos los mapas en estos ficheros? Diseña un mapa nuevo, guárdalo y ejecútalo en el juego.

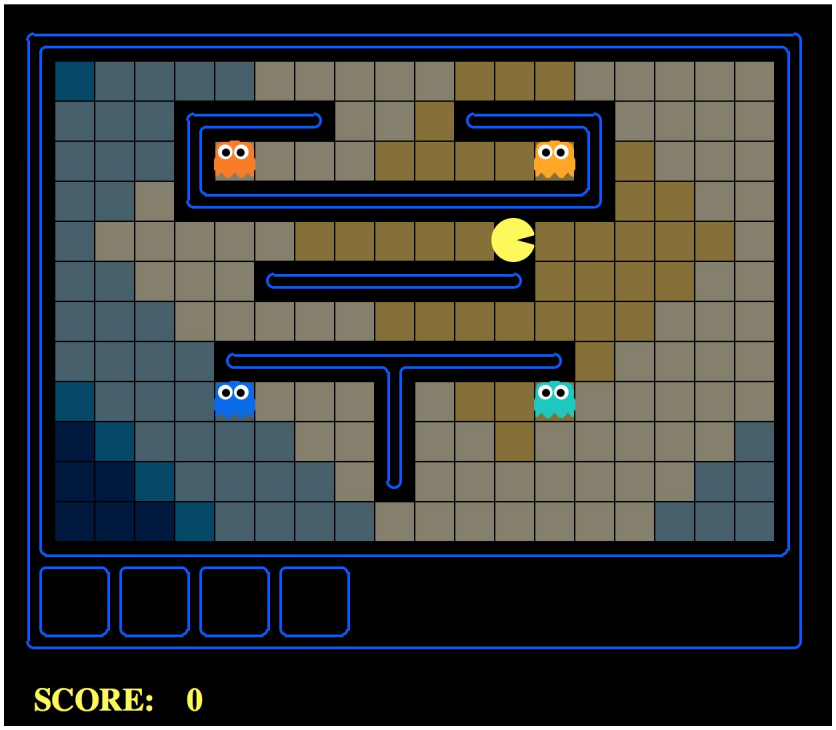
Están definidos por caracteres en archivos de texto con la extensión .lay. Cada carácter representa una casilla en el mapa.

- **% - Paredes**. Son casillas que PacMan no puede atravesar
- **G** - Representa la posición inicial de los fantasmas.
- **P** - Representa la posición inicial de PacMan.
- **.** - Casilla que PacMan puede comer. En esta implementación estas casillas no son tenidas en cuenta.
- **o** - Casilla que PacMan puede comer. En esta implementación estas casillas no son tenidas en cuenta.
- Además, si el formato de la penúltima fila del mapa es del tipo

  % % % % % % %%%%%%%%%%%%%%%%

  en el juego se verá qué fantasmas han sido comidos por PacMan en la parte inferior del mapa.

El nuevo mapa creado por nosotros es el siguiente:



Pregunta 4

Analiza el fichero game.py. ¿Qué información ofrece este código sobre el estado del juego en cada turno? De esta información, ¿cuál crees que podría ser más útil para decidir automáticamente qué tiene que hacer PacMan?

En *game.py* podemos encontrar la clase `GameStateData`. Esta clase contiene cierta información sobre el estado del juego, como puede ser la puntuación, si la partida se ha ganado o perdido, la cantidad de comida que se ha comido, las cápsulas que se han comido, si nueva comida ha sido añadida, e información sobre el estado de los agentes.

Además, en la clase `Agent` se pueden ver datos sobre los agentes, tales como su configuración (posición y dirección), o si el agente es PacMan o un fantasma.

La clase `Game` contiene referencias a los agentes y a la forma en la que el juego se muestra en la pantalla, así como el historial de movimientos realizados.

De la información que se puede encontrar en este archivo, la más relevante para PacMan podría ser aquella relaccionada con el estado de los agentes (fantasmas), aunque parte de esta información (por ejemplo, la posición) no está permitida obtenerla directamente y debe ser inferida a partir de otros parámetros. Además, la propia dirección y posición de PacMan es también necesaria, tal y como se ha comentado en la pregunta 2.

Pregunta 5

Programa una función que imprima en un fichero de texto toda la información del estado del PacMan. Esta función servirá como una primera versión de la fase de extracción de características que será imprescindible en las siguientes prácticas.

- *Por cada turno de juego, se debe guardar una línea con todos los datos concatenados del estado del juego que se calculan por defecto, separados por el carácter coma (,).*
- *Además, cada vez que se inicie una nueva partida o se abra el juego de nuevo, las nuevas líneas deben guardarse debajo de las antiguas. Es decir, que no se debe reiniciar el fichero de texto al empezar una nueva partida.*
- *Por tanto, el fichero de texto resultante tendrá que tener tantas líneas como turnos se hayan jugado en todas las partidas.*

El objetivo de este apartado era crear una función que sacase a un archivo de texto inforamción relevante acerca del estado de la partida para que nuestro agente pueda aprender de estos datos en un futuro. El formato del archivo se corresponde con una línea por cada turno de la partida en la que los valores se encuentran separados por una coma.

Para abrir (o crear en el caso de que no exista) un archivo en Python y añadir nuevas líneas sin borrar las ya existentes, debemos hacerlo de la siguiente manera:

```
f = open('filename', 'a+')
```

Estos son los datos que sacamos, separados por comas, en este orden:

- **Fantasmas que están vivos**: Se representan por booleanos, siendo True, que el fantasma que se encuentra en la posición i está vivo, y False, no. El primer dato se corresponde con un placeholder para PacMan cuyo valor será siempre False.
- **Distancia de PacMan a cada uno de los fantasmas**: Esta distancia contendrá cierto ruido y no será del todo precisa.
- **Puntuación de la partida**: La puntuación de la partida en cada turno se obtiene por si en un futuro el objetivo de la práctica es maximizar este valor.
- **Posición de PacMan**: Obtenemos la posición de PacMan para saber en qué lugar del mapa nos encontramos.
- **Dirección de PacMan**: La dirección de PacMan también se obtiene para saber hacia dónde nos estamos moviendo (indica la última acción tomada).
- Además, los últimos datos recogidos se corresponden con 8 valores booleanos, los cuatro primeros indican si alguna de las **celdas adyacentes** a las que PacMan podría moverse se corresponden con un movimiento ilegal (muro o límite del mapa) y, los otros cuatro, si alguna de estas cuatro celdas contiene comida.

Pregunta 6

Implementa manualmente un comportamiento para el PacMan. Para ello se debe modificar la clase del agente GreedyBustersAgent que se encuentra dentro del fichero bustersAgents.py. Esta clase es sólo una plantilla que sirve como punto de partida. PacMan debe perseguir y comerse a todos los fantasmas de la pantalla.

En esta sección, comenzamos implementando un agente basándonos en las indicaciones del código fuente. PacMan elegía como objetivo al fantasma más cercano fuese cual fuese la posición del resto. Basaba su decisión en la información dada por un array de Counters `livingGhostPositionDistributions` , que contiene pares del tipo (x,y: probabilidad de estar ahí) para cada uno de los fantasmas. Su funcionamiento era correcto basado en los datos que recibía, pero estos eran ciegos por lo que el resultado era similar al de un agente aleatorio.

Tras las aclaraciones posteriores del profesor de prácticas en las que se nos indicaba que los datos a tener en cuenta no eran los dados por el código sino la tupla conteniendo las distancias a los fantasmas (atributo de `GameState.data`, de donde también sacamos la dirección de PacMan que nos indica su última acción), nos pusimos manos a la obra para diseñar un nuevo agente basándonos tan sólo en esa información.

Rápidamente descubrimos que la información de un turno aislado no aporta información alguna, pues si un fantasma está "a cinco posiciones", cualquier acción (norte, sur, este u oeste) tiene a priori las mismas posibilidades de éxito. De esta forma, tratamos de buscar más información a partir de los mismos datos.

Llegamos a la conclusión de que, almacenando como atributo de nuestro agente la anterior tupla de distancias podríamos evaluar en el siguiente paso si vamos o no en buena dirección. Esto nos permite decidir entre dos opciones:

- **Seguir avanzando** (repetir la última acción): si nos estamos acercando al objetivo o si éste no se ha alejado (porque, si no se aleja, significa que nos movemos en su misma dirección y por tanto la acción tomada era la correcta).
- **Cambiar de dirección** (cambiar a una acción aleatoria): si nos alejamos o es ilegal seguir avanzando.

La elección de seguir avanzando o cambiar de dirección la toma el agente comparando los **mínimos** de cada una de las tuplas (actual y previa). De esta forma, aunque se aleje del fantasma al que perseguíamos, si PacMan se acerca a otro fantasma y este queda más cerca de él que antes, cambiará de objetivo automáticamente y seguirá yendo en la misma dirección.

Los movimientos efectuados por este nuevo agente parecen *tener más sentido* que los del primero cuando le observas jugar mostrando a los fantasmas, y el rendimiento de éste mejora de forma observable. Aún así, en nuestro equipo confiamos en que, si se aplicasen técnicas de aprendizaje automático, se podría mejorar notablemente la efectividad del agente.

Pregunta 7

El agente programado en el ejercicio anterior no utiliza ninguna técnica de aprendizaje automático. ¿Qué ventajas crees que puede tener el aprendizaje automático para controlar a PacMan?

Aplicando técnicas de aprendizaje automático el rendimiento de PacMan podría mejorarse notablemente consiguiendo que nuestro agente tomase mejores decisiones para comer antes a los fantasmas.

Estas técnicas podrían basar su "experiencia", esto es, los datos de los que sacaría conocimiento, en los movimientos anteriores etiquetados como *buenos* o *malos* dependiendo de si acercaron o alejaron a PacMan de su objetivo partiendo de una situación inicial concreta (resto de parámetros).

De esta forma, dada la situación actual el agente tomaría unas u otras decisiones fundamentadas sobre la experiencia previa, la cual iría aumentando según jugase más partidas.