Alberto Sánchez Amo - 152871
Luis Ángel Rodríguez Viñal - 154186
Grupo P102

Computación Inteligente

Práctica 1: Berkley Pacman

Funcionalidad desarrollada

El código se encuentra comentado para facilitar la comprensión.

La práctica implementa las siguientes cuestiones del proyecto:

- **Question 1**: DepthFirstSearch DFS - Implementación correcta

  Usamos una pila donde guardamos información acerca de los nodos que hemos visitado.

- **Question 2**: BreadthFirstSearch BFS - Implementación correcta

  La metodología aplicada ahora será buscar en amplitud, por lo que en lugar de usar una pila usamos una cola.

- **Question 3**: Dijsktra - Implementación correcta

  Como este algoritmo considera el coste de cada nodo como un valor a tener como prioridad (el de menor coste primero) usamos una cola con prioridades, donde la prioridad será el coste.

- **Question 4**: A* - Implementación correcta

  Siguiendo la misma metodología que en la Question 3 pero añadiendo al coste de cada nodo la heurística de dicho nodo.

- **Question 5**: Modelar un estado - Implementación correcta

  La definición de nuestro estado state tiene la siguiente estructura:

  - En el state[0] definimos la posición en la que se encuentra Pacman
  - En el state[1] guardamos una tupla inicialmente vacía, en la que vamos añadiendo las posiciones que son visitadas (en este caso iremos añadiendo esquinas).
    Por ejemplo:

    self.state[1] = [(right,top)]

    En este caso nuestro estado state[1] indica que hemos visitado una esquina, la esquina de arriba a la derecha.
  - El goal por tanto contendrá todas las esquinas. En nuestro caso:

    self.goal = [(1,1), (1,top), (right, 1), (right, top)]

- **Question 6**: Definición de una función heurística - Implementación no óptima

  Nuestra función heurística consiste en valorar la distancia Manhattan que hay entre la posición actual y una esquina. Devolvemos el valor del coste máximo del camino.

  La heurística es consistente porque el valor nunca excederá el coste (como mucho lo igualará):

  $$h(N) <= c(N, A) + h(S)$$

  donde N es un nodo, S su sucesor y A la acción que hacemos desde N.

  Una acción válida tiene coste 1. La distancia de Manhattan entre dos posiciones se calcula de manera que el camino que se coge usa valores o sólo verticales (suma o resta 1 valor al eje Y) o sólo horizontales (suma o resta 1 valor al eje X).

Problemas a lo largo del desarrollo

Los principales problemas que hemos tenido han sido debido a problemas con la familiarización del lenguaje Python, especialmente cuando intentamos guardar los valores del coste o las acciones dentro de una misma estructura (stack,pila,cola) hemos tenido confusión a la hora de gestionar las tuplas y las listas dentro de este. Finalmente hemos preferido separar ambas listas en diferentes estructuras (nodos y acciones) para evitar confusiones, el funcionamiento y resultado final tanto en una como en dos estructuras es el mismo.

Otro problema que hemos tenido (solo en un portátil) ha sido la imposibilidad de compilar y usar la librería Twinker que usa este proyecto, y descargando dichas librerías de nuevo y volviendo a instalarlas no ha servido.

Conocimientos adquiridos

Hemos comprendido el funcionamiento y el comportamiento de los algoritmos de búsqueda, primeramente con algoritmos más simples como BFS y DFS, seguidamente de algoritmos que utilizan métodos más avanzados como es el caso de A* y el uso de una función heurística para estimar el coste.

A su vez, hemos podido comprobar de primera mano como estos algoritmos son de gran utilidad, donde el claro ejemplo lo hemos podido ver en el juego Pacman. También hemos podido ver un modelo de estados en acción.

Globalmente hemos conseguido una familiarización con Python a la vez que con estos algoritmos.