Informe de Inteligencia Artificial

Primer Proyecto: Búsqueda

Autor: Masiel Villalba Carmenate

Grupo: C-512

El proyecto consta de 8 preguntas las cuales han servido para comprender los problemas de búsqueda y esclarecer dudas emergentes.

Pregunta 1

Para la implementación del DFS se definió la clase *Node* con las propiedades *state, parent, action\_since\_parent y way\_cost*(cantidad de nodos en el camino desde el origen para alcanzarlo)Luego se respeta la lógica del DFS: se tiene una pila y en cada iteración se extrae de ella el objeto tipo *Node* que está en su tope, se expande(si su estado no es objetivo, en cuyo caso el ciclo termina) y se inserta en la lista de nodos expandidos, posteriormente se recorren sus sucesores y por cada uno, si no se ha expandido, se inserta en la pila. El camino(la lista de acciones) se construye a partir del último nodo que se expanda, con el método build\_path\_actions de *Node.*

Preguntas 2-4

Inicialmente se esperaba implementar todos los algoritmos de búsqueda con un solo método pero luego de analizar los detalles de estos, sale a relucir la peculiaridad del DFS de que los “nodos” expandidos no se pueden marcar como tal en el mismo momento que se visitan, puesto que esto detendría la búsqueda en profundidad por causa de algún ancestro que este más lejos en la pila. Pero por suerte, BFS, UCS y A\* son idénticos según este criterio y por eso han podido implementarse con la misma función: generic\_search\_function. En este caso la lógica de *Node* explicada *arriba* se ha expresado con 3 diccionarios: parents, distances y action\_since\_parent. Cada uno al indexar por el estado retornan, respectivamente, el estado padre, el costo de alcanzar el estado y la acción ejecutada desde el padre para visitarlo. Se utiliza una PriorityQueue como estructura de datos y cada estado se inserta en ella con el costo del camino hasta él, sumado al valor de la heurística. Así, las rutinas para BFS y UCS, no tienen diferencias, en las dos se utiliza la heurística nula y A\* funciona correctamente con la heurística que se le pase como parámetro.

Pregunta 5

La codificación elegida para los estados es una tupla binaria donde la primera componete expresa la posición de Pacman en el estado y la segunda, es una tupla con las esquinas que ya se han visitado. De esta manera, getStartState retorna (initial\_pacman\_position,()), isGoalState chequea que la segunda componente del estado tenga tamaño 4 y getSuccessors retorna las tuplas tipo (nuevo\_estado, acción,costo) donde al nuevo estado se le añade como visitada su primera componente si es una esquina, y se mantienen las esquinas visitadas de su padre.

Pregunta 6

Para resolver el valor de la heurística se ha definido el problema Relaxed\_Food\_Problem, para el cual cada estado es una tupla que contiene en su primera componete la posición actual de Pacman y en la segunda las esquinas que le falta por visitar, pero sus sucesores no se hallan desplazando a Pacman en solo un paso, sino directamente a alguna de las casillas que debe visitar y se toma como costo la distancia de Manhattan de hacer el salto. Así la solución de Relaxed\_Food\_Problem mediante UCS es un camino de tuplas iniciando por la casilla en que está Pacman. Tal camino simboliza el recorrido de “menor costo”(porque en realidad obvia los obstáculos) que se debe hacer para visitar las esquinas y el costo de este es el valor que retorna la heurística. Se dice que será admisible porque si el tablero no tuviese muros, devolvería entonces la cantidad de pasos que verdaderamente habría que dar para llegar a la solución, pero en otro caso, la distancia de Manhattan siempre será menor o igual que el costo de trasladarse de un punto a otro en el laberinto.

Pregunta 7

Se le añade al diccionario *heuristicInfo* de *FoodSearchProblem* la propiedad ‘nearest’ que almacena la posición de la casilla con comida más cercana(realmente) en el laberinto a Pacman. En cada llamado a foodHeuristic se verifica que esta posición aún contenga comida(de lo contrario se recalcula) y finalmente se retorna la distancia de Manhattan de Pacman a esa casilla sumado a la cantidad de comida que falta por consumir menos 1.

Pregunta 8

Para solucionar este ejercicio solo se ha definido *isGoalState* de *AnyFoodSearchProblem* y se retorna la solución del problema hallada con BFS.