COMPUTACIÓN INTELIGENTE

Búsqueda en Pacman

Joan Grau 172578

Ignacio Estaún 173987

¿Qué funciona?

Las preguntas obligatorias Q1-Q6, y las opcionales Q7-Q8.

¿Qué problemas encontraron?

Manca de formación de estructuras de datos de programación, poca información en el guión de la práctica, y los recursos que se nos propusieron fueron algo escasos, pero creemos que es debido a que nosotros somos estudiantes de Audiovisuales y no de Informática.

Q1. Is the exploration order what you would have expected? Does Pacman actually go to all the explored squares on his way to the goal? Is this a least cost solution? If not, think about what depth-first search is doing wrong.

- Sí, sabemos que el DFS lo que hace es expandir un nodo y uno de sus hijos hasta llegar al final de la rama (que es cuando no quedan nodos por visitar en este camino o que hayamos llegado a la meta).
- No se exploran todas las casillas del juego cómo es de esperar, ya que el modelo de Pila lo que hace es sacar las últimas entradas que le hemos puesto, de modo que a más nodos explorados en una rama, y por tanto más nodos puestos en la pila, menos probabilidad tenemos de llegar a uno de los primeros nodos puestos en la pila.
- Probablemente no sea la función de menor coste, ya que sólo explora un camino usando un algoritmo que simplemente va avanzando hasta que llega a la meta o a un límite, sin tener en cuenta otros factores como el coste o otros caminos.

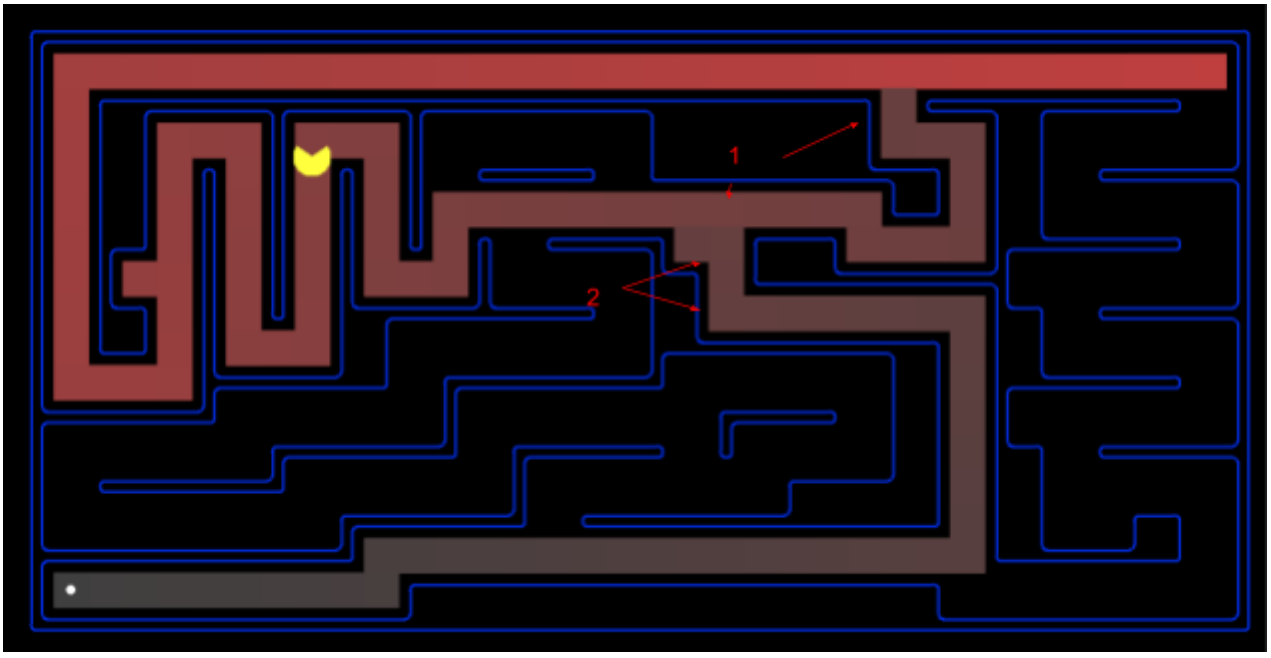

Figura 1: vemos cómo se ha explorado una rama desde el principio. Vemos que el algoritmo ha intentado de buscar la solución volviendo en un punto por el camino que ya había pasado (1), y ha rectificado siguiendo desde la ramificación más próxima desde el final (2) (propiedad de la pila)

Q2. Does BFS find a least cost solution? If not, check your implementation.

Sí, lo que hace el BFS es buscar en “ramas”, o más simplemente podemos decir que va buscando por niveles, de manera que todos los caminos que busca los va creando nivel por nivel. De este modo, el primer nivel que encuentre, será el más corto, asumiendo que el coste por cada casilla es uniforme.

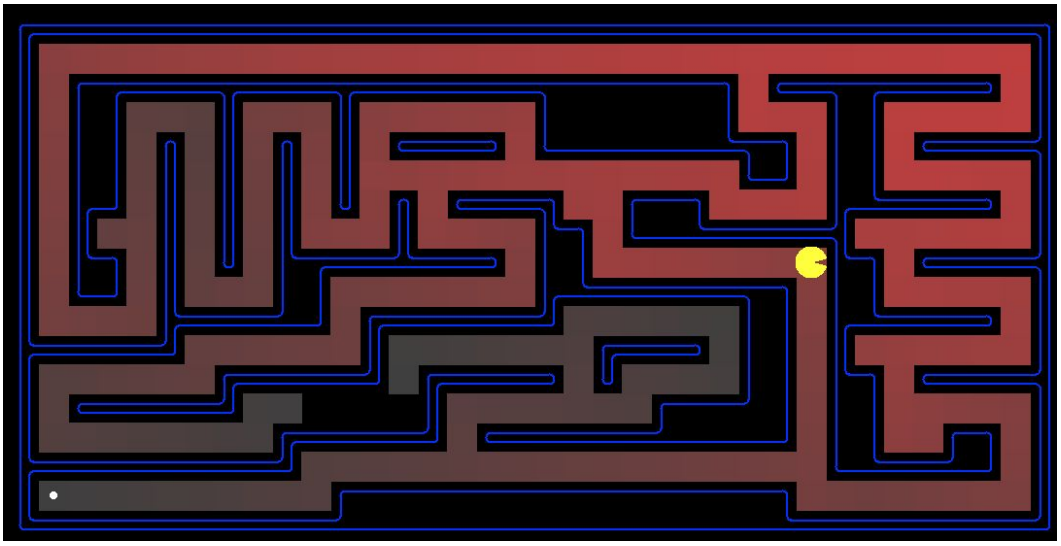

Figura 2: podemos ver que a diferencia de el anterior algoritmo, este ha tenido en cuenta muchas de las ramificaciones del laberinto. Vemos que en el centro hay un trozo que no se ha explorado, debido a que antes que se explorarse ya se ha encontrado un camino, que se asume que es el más corto.

Q4. What happens on openMaze for the various search strategies?

Search Algorithm	Cost	Time	Nodes Expanded	Score
DFS	298	0.1	577	212
BFS	***54***	0.1	682	***456***
UCS	***54***	0.2	682	***456***
A*	***54***	0.2	***554***	***456***

- Cómo podemos ver, el único algoritmo que no ha encontrado la solución “óptima” es el DFS, el cual ha llegado a la meta pero a un coste muy grande.
- El resto de algoritmos se diferencian entre sí sólo en el número de nodos expandidos. Entre BFS y UCS no hay diferencia, ya que en la función de coste uniforme estamos usando como heurística una función que no calcula nada (*nullHeuristic*), de modo que el resultado es esperado.
- La función A* si usa una heurística (Manhattan distance), y se nota que ayudado ya que ha necesitado expandir 100 nodos menos que los otros dos algoritmos.

Q5, Q6 y Q7

- Para estos tres puntos, hemos basado nuestro algoritmo en utilizar la distancia de Manhattan y adaptándola al problema para crear heurísticas en el caso de la Q5 y Q6, y en el caso de la Q7 cómo algoritmo de búsqueda.
- Lo que hemos hecho en las esquinas por ejemplo, es buscar qué esquinas faltaban ser encontradas, y luego iterar por cada una de ellas y buscar la distancia de cada una hasta la posición actual. Nos quedamos con la esquina con mayor coste, sumamos su coste al ya calculado, lo borramos de la lista y actualizamos la posición de este a la esquina. Luego hacemos lo mismo por las esquinas que nos falten. De este modo, tendremos la distancia mínima, calculada con Manhattan Dist., de ir de esquina a esquina desde la posición actual hacia todos los nodos. Es un método un poco costoso pero nos ha dado buenos resultados.

Además, en el Q6 también hemos implementado un algoritmo para asegurar que no entramos en callejones sin salida, de modo que no perdamos tiempo entrando en ellos, lo hacemos usando una profundidad arbitraria y usando parámetros que usamos en la Q5.