Practica 1 Inteligencia Artificial: Marcos Alonso Arnanz

1. A la hora de implementar el algoritmo de búsqueda en profundidad (DFS), se ha optado por la utilización de una pila como estructura de datos, dado las características propias del algoritmo (LIFO) estudiadas en la teoría.
2. A la hora de implementar la búsqueda en amplitud (BFS), se ha optado por la utilización de una cola como estructura de datos, dada las características propias del algoritmo (FIFO) estudiadas en la teoría.
3. A la hora de implementar al algoritmo de variación de la función de coste (UCS), se ha optado por la utilización de una cola con prioridad como estructura de datos.
4. A la hora de implementar el algoritmo A* se ha optado por una cola con prioridad como estructura de datos.

La heurística de los cuatro primeros ejercicios es esencialmente la misma. Se inicial la estructura de datos escogida con un nodo correspondiente al estado inicial. Posteriormente, siempre y cuando la estructura de datos no esté vacía -si se vacía antes de encontrar una solución entonces no ha solución al problema y se retorna una lista vacía dejando a PacMan quieto- se comprueba si el nodo actual es la solución al problema. De no ser así se comprueba sus sucesores (siempre y cuando no hayan sido visitados) y se calcula el coste de ser preciso.

Una vez se encuentra el nodo de destino se devuelve una lista con las instrucciones de movimiento para PacMan.

6. A la hora de desarrollar la heurística para el problema se las esquinas, se ha optado por calcular la distancia de Manhattan dado que la más optima para los problemas de rejilla en la que solo se permite el movimiento en cuatro direcciones.
7. Si bien la distancia de Manhattan parecía la más indicada para este problema las aproximaciones efectuadas con el algoritmo desarrollado para el apartado anterior (adaptado a las particularidades) del mismo aportaban un resultado entorno a los 9.000 nodos expandidos. Mediante algunos ajustes se ha conseguido reducir el numero de nodos expandidos hasta los 8.000 si bien no era suficiente para obtener los resultados necesarios para completar el ejercicio.
   En consecuencia, se ha optado por utilizar la función **mazeDistance** incluida en el código original que calcula la distancia entre dos puntos haciendo uso de las paredes. Con dicha aproximación se ha conseguido resolver el ejercicio expandiendo un total de 4.137 puntos con un tiempo de aproximadamente 50 segundos.
   Como observación necesaria indicar, que, si bien esta aproximación expande aproximadamente la mitad de los puntos que las anteriormente realizadas, esta toma entorno a 5 veces más tiempo para su cálculo – las primeras aproximaciones tomaban en torno a 10 segundos para su ejecución-.
8. En el último apartado solo ha sido necesario retornar el valor correspondiente (*True* o *False*) para el estado actual del juego en la función **isGoalState**. Para la función **findPathToClosestDot** tan solo ha sido necesario indicar el algoritmo deseado. No se han apreciado cambios entre el uso de A*, UCS y BFS, dando todos un coste de 250 en tiempo inferior a 1 segundo