UNIVERSIDAD
POLITÉCNICA
DE MADRID

Práctica 3: Representaciones del espacio y planificación

Cesar García Cabeza

Curso 2020-21

Fecha: Enero de 2021

Asignatura:
Robots Autónomos

Profesor:
D. Nik Swoboda

Índice

Cesar García Cabeza

1. Material entregado

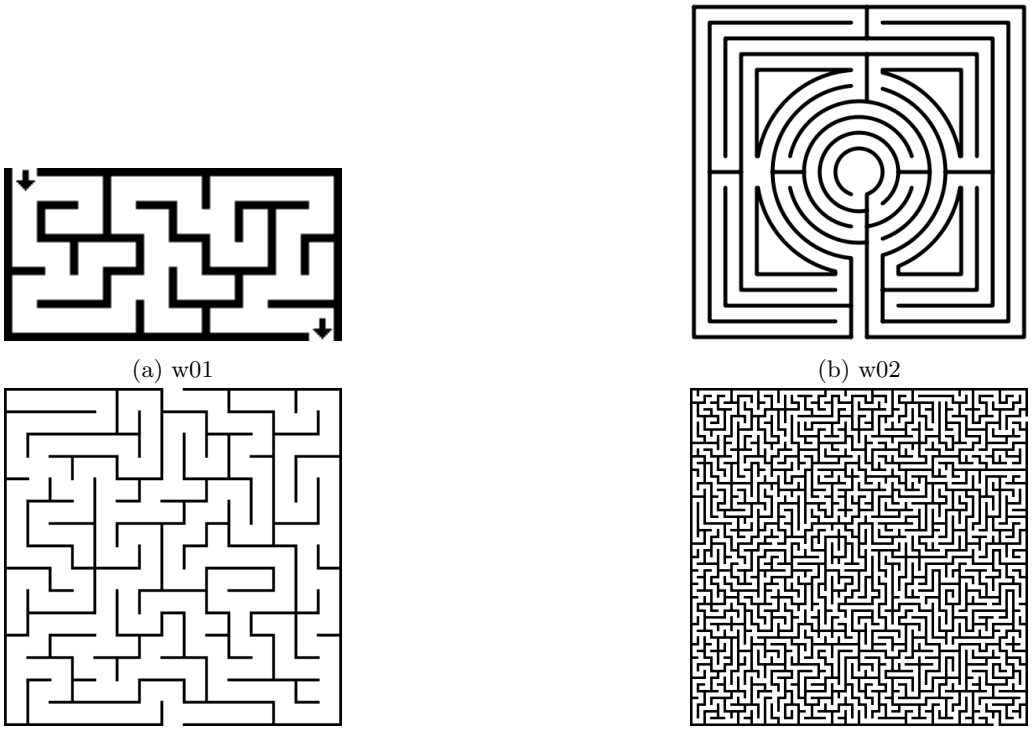
Para esta práctica se ha entregado un fichero comprimido de nombre *Garcia-Cabeza_RA3.zip*. En él podrá encontrar lo siguiente:

- Una carpeta **worlds**: en ella se encuentran 6 escenarios que se han usado para probar nuestra implementación.
- Una carpeta **outputs**: en esa carpeta es donde se guardan los resultados. Se entregan varios resultados de varios puntos de comienzo y fin de varios algoritmos para varios mundos.
- Una carpeta **src**: en ella se encuentra el código implementado. Contiene un fichero **worlds.py** encargado de la lectura, creación y manejo del mundo, un fichero **algorithms.py** con las implementaciones propias de los algoritmos y un fichero **planner.py** que es el encargado de ejecutar el planificador.

2. Mundos

Los mundos que se entregan en esta práctica son los siguientes:

- W01, w02, w03 y w04 son cuatro laberintos, a cada cual más grande y complejo.



(a) w01 (b) w02

(c) w03 (d) w04

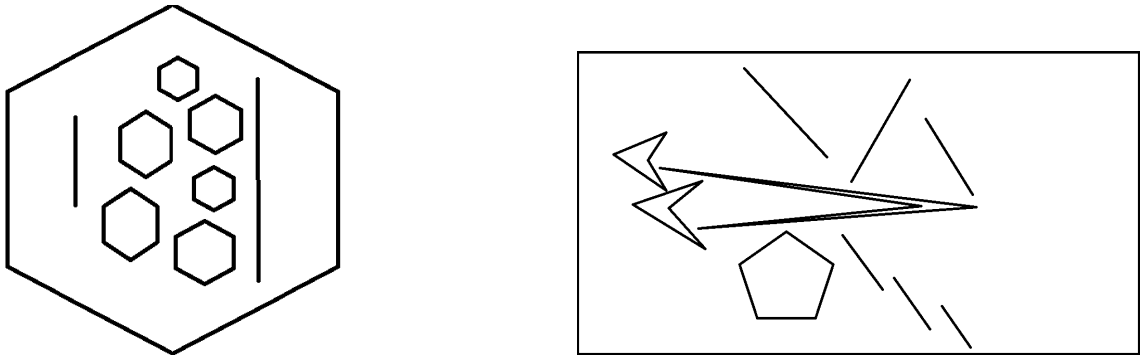
Figura 1: Laberintos

- W05 y w06 son dos escenarios de "mundo libre". (Véanse en la Figura 2)

3. Algoritmos

Los algoritmos que se han empleado en la práctica han sido:

1. **Value Iteration**: algoritmo explicado en clase que consiste en una primera fase en la que se rellena el tablero con las distancias de cada celda a la meta y una segunda fase en la que se empieza en el



(a) w05 (b) w06

Figura 2: Mundos libre

comienzo y se va cogiendo siempre el camino de menor valor. Se ha implementado de tal forma que una vez se detecta que se ha llegado al comienzo se para de rellenar el tablero con las distancias, haciéndolo más eficiente.

2. **Dijkstra** o Breadth First Search: el cual realiza una búsqueda en anchura.
3. **Depth First Search**: que realiza una búsqueda en profundidad.
4. **Best First Search**: realiza una búsqueda heurística, escogiendo siempre el mejor nodo posible.
5. **A***: realiza una búsqueda mezclando el coste real de un movimiento con una heurística.

Destacar que todos estos algoritmos son completos, es decir, obtienen siempre una solución. Además, se han implementado todos los algoritmos por cuenta propia.

Se podría haber optado por implementar otro tipo de algoritmos como mapas de caminos probabilísticos o los algoritmos de muestreo; sin embargo, al centrarnos en laberintos, pensamos que estos enfoques no iban a funcionar tan bien como Value Iteration u otros algoritmos de búsqueda tradicionales.

4. Intrucciones de uso

A continuación se detallan los pasos necesarios para poder ejecutar nuestro proyecto **(en Windows)**:

1. Deberá tener instalado Python.
2. Abra la terminal y entre en la carpeta /García-Cabeza_RA3/src.
3. Instale las librerías necesarias ejecutando *pip install -r requirements.txt*. En caso de tener algún problema instalando las librería de este modo, deberá instalarlas una a una mediante *pip install librería*.
4. Una vez instaladas las librerías deberá ejecutar planner.py.
   - El primer argumento corresponde con el mundo que quiere probar.
   - Después, mediante -a, podrá escoger qué algoritmo ejecutar de entre: "dijkstra", "dfs", "best", "a-star", "valit".

   Se adjuntan varios ejemplos de posibles ejecuciones:

   - **Completa**: *python planner.py ..\worlds\w01.png -a dijkstra -a dfs -a best -a a-start -a valit*
   - **Parcial**: *python planner.py ..\worlds\w03.png -a valit*
5. Se le pedirá escoger dos puntos en la imagen: haga click en dos posiciones distintas y cierre la imagen. (Figura 3)

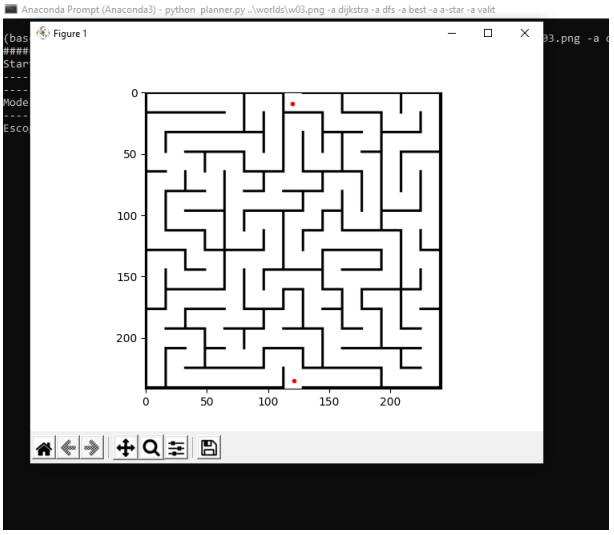


Figura 3: Cierre la imagen una vez seleccionados dos puntos.

6. Se le irá mostrando en pantalla información sobre qué algoritmo se está ejecutando, tiempo de ejecución...

7. Finalmente se le indicará que ha finalizado la ejecución y que puede encontrar los resultados en la carpeta outputs. (Figura 4)

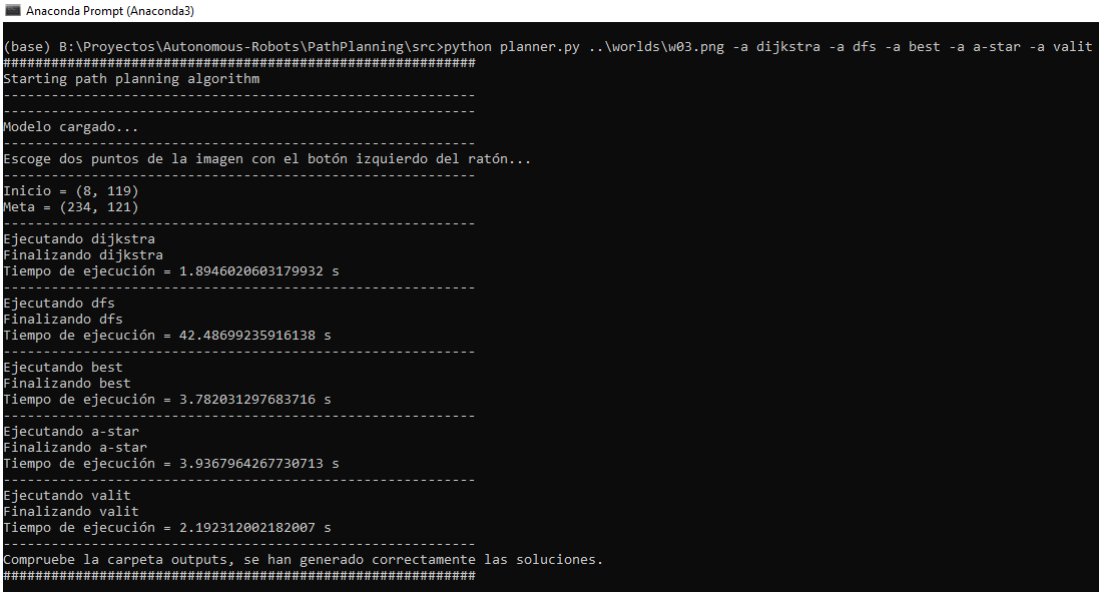


Figura 4: Información mostrada.