

## INTELIGENCIA ARTIFICIAL

### CURSO 2023-24

#### PRÁCTICA 2: Repertorio de preguntas para la autoevaluación de la práctica 2.

APELLIDOS Y NOMBRE	Morro Tabares, Isabel		
GRUPO TEORÍA	DGIIM	GRUPO PRÁCTICAS	2

#### Instrucciones iniciales

En este formulario se proponen preguntas que tienen que ver con ejecuciones concretas del software desarrollado por los estudiantes. También aparecen preguntas que requieren breves explicaciones relativas a cómo el estudiante ha hecho algunas partes de esa implementación y qué cosas ha tenido en cuenta.

En las preguntas relativas al funcionamiento del software del alumno, estas se expresan haciendo uso de la versión de invocación en línea de comandos cuya sintaxis se puede consultar en el guion de la práctica.

El estudiante debe poner en los recuadros la información que se solicita.

En los casos que se solicita una captura de pantalla (**ScreenShot**), extraer la imagen de la ejecución concreta pedida donde aparezca la línea de puntos que marca el camino (justo en el instante en el que se construye obtiene el plan). Además, en dicha captura debe aparecer al menos el nombre del alumno. Ejemplos de imágenes se pueden encontrar en [Imagen1](#) y en [Imagen2](#).

#### Consideraciones importantes:

- Antes de empezar a rellenar el cuestionario, actualiza el código de la práctica con los cambios más recientes. Recuerda que puedes hacerlo o bien realizando **git pull upstream main** si has seguido las instrucciones para enlazar el repositorio con el de la asignatura, o bien descargando desde el enlace de GitHub el zip correspondiente, y sustituyendo los ficheros **jugador.cpp** o **jugador.hpp** por los tuyos.
- Si en alguna ejecución consideras que tu agente se ha visto perjudicado puedes añadirlo a los comentarios en el comentario final (al final del documento).

**Enumera los niveles presentados en su práctica (Nivel 0, Nivel 1, Nivel 2, Nivel 3, Nivel 4):**

Nivel 0, Nivel 1, Nivel 2, Nivel 3, Nivel 4
---

Nivel 0-Anchura para el agente jugador

(a) Rellena los datos de la tabla con el resultado de aplicar `./practica2 mapas/mapa30.map 1 0 5 5 2 10 10 4 12 5`

ScreenShot	
Instantes de simulación no consumidos	2994
Tiempo Consumido	0.001642
Nivel Final de batería	2987

## Nivel 1-Anchura para el agente colaborador

- (a) ¿En qué se diferencia desde el punto de vista de la implementación el algoritmo que has usado en este nivel en relación al del nivel 0? Enumera los cambios y describe brevemente cada uno de ellos.

El objetivo ya no es llevar al jugador a la casilla final, si no que queremos llevar al colaborador a esta. Por ello, ha sido necesario realizar los siguientes cambios:

- La definición del estado del nivel 1, el operador == compara tanto la ubicación completa del jugador como del colaborador, además de la última orden dada al colaborador. Asimismo, el operador < del nodo del nivel 1 compara fila, columna y brújula del jugador y colaborador, además de la última orden mandada a este último.
- Para mandar órdenes al colaborador es necesario que el jugador sea capaz de visualizarlo en el momento en el que se da la orden, por ello he creado el método colaboradorAVista. Este, dependiendo de la dirección a la que mira el jugador, calcula a partir de su posición las 15 casillas que puede ver (en la 0 está el mismo) y compara si alguno de esos valores coincide con la posición del colaborador, en cuyo caso devolverá true.
- En el algoritmo de anchura, en este nivel se añaden los nodos hijos de las acciones del colaborador, y se pone como objetivo que sea la posición del colaborador la que coincida con la posición del objetivo final.
- La mayoría de los cambios se encuentran en el apply, donde, si la acción a realizar es del jugador, será válida solo si la última orden del colaborador puede llevarse a cabo. En caso contrario, si la acción a realizar es del colaborador, será válida si este se encuentra en la vista del jugador, además de las comprobaciones básicas como que sea una casilla transitable, o que la casilla a la que se accede no sea en la que se encuentra el jugador. Se debe tener en cuenta que el jugador se mueve primero, luego si el jugador camina, el colaborador no puede acceder a la casilla a donde se ha movido el jugador, pero si a donde inicio el movimiento.

(b) Rellena los datos de la tabla con el resultado de aplicar:  
**./practica2 mapas/mapa30.map 1 1 9 3 0 7 3 6 3 3**

ScreenShot		
	Instantes de simulación no consumidos	2994
	Tiempo Consumido	0.001461
	Nivel Final de Batería	2988

(c) Rellena los datos de la tabla con el resultado de aplicar:  
**./practica2 mapas/mapa30.map 1 1 24 12 6 18 13 6 18 15**

ScreenShot	
<b>Instantes de simulación no consumidos</b>	2976
<b>Tiempo Consumido</b>	7.75774
<b>Nivel Final de Batería</b>	2957

## Nivel 2-Dijkstra para el agente jugador

- (a) ¿Qué es propio de este nivel que no tuviste que tener en cuenta en los niveles anteriores? Enumera los cambios y describe brevemente cada uno de ellos y que han implicado en la implementación.

En este nivel se busca que el jugador llegue al objetivo con el mayor nivel de batería posible, o del mismo modo, con el menor gasto posible. Para ello se han llevado a cabo los siguientes cambios:

- La definición del estado del nivel 2 ahora incluye un booleano tanto para saber si el jugador tiene las zapatillas, como otro para saber si el jugador tiene el bikini. El operador `==` del estado compara la ubicación del jugador y si tiene o no bikini o zapatillas. Ahora bien, la definición del nodo del nivel 2, incluye una variable llamada “coste” que irá almacenando el coste de la secuencia de acciones de nodo. Cabe resaltar el operador `<` del nodo del nivel 2, pues en este se considera un nodo menor aquel que mayor coste tenga, esto se comentará más adelante cuando se explique el algoritmo.
- He definido un método llamado `ObtencionElementos`, que comprueba si en la casilla donde se encuentra el jugador obtiene bikini o zapatillas. Teniendo en cuenta que tan solo puede tener uno de estos dos objetos y que siempre se queda con el último encontrado. Además, también he definido un método llamado `ValorCoste`, que devuelve el coste de realizar la acción pasada como parámetro. Este coste dependerá de los elementos que tenga el agente, el tipo de suelo y la acción a realizar.
- La función `apply` es prácticamente igual que en nivel 0. Sus ligeros cambios son la suma del coste de la acción (siempre que esta se pueda realizar) y la comprobación de la obtención de elementos al final de esta.
- Por último, tenemos el algoritmo de costo uniforme llamado `MinimoCaminoJugador`. Al comienzo de este se comprueba si en la casilla donde comienza el jugador tiene la opción de conseguir alguno de los elementos. De resto se crean los nodos hijo de las acciones propias del jugador. Cabe resaltar que, al contrario que con la búsqueda de anchura, ya no se comprueba en cada nodo hijo creado si este es solución, si no que la solución debe encontrarse en cerrados. En este nivel, manejamos un set de estados y una cola con prioridad de nodos, pues así comparamos todo lo relacionado con información del jugador en los estados, y en la cola tan solo el coste. Esta cola se ordena de menor a mayor, es por ello que el operador `<` fue definido como explicamos anteriormente, pues nos interesa que se tomen aquellos nodos con mayor coste como “menores”, por ello en el `if` del final del bucle `while` hacemos `frontier.pop()` y nos quedamos con el último nodo de la cola.

- (b) ¿Has incluido dentro del algoritmo de búsqueda usado en este nivel que, si pasas por una casilla que da las zapatillas o el bikini, considere en todos los estados descendientes de él, el colaborador o el jugador tiene las zapatillas y/o el bikini? En caso afirmativo, explicar brevemente cómo.

El colaborador no lo he tenido en cuenta, pues no interesa en este nivel. De cara al jugador, como he mencionado en la pregunta anterior, es en el apply donde se comprueba si consigue o no bikini o zapatillas, modificando el booleano correspondiente del nodo del nivel 2.

Para cada acción del jugador, en primer lugar, se crea un nodo que se iguala a `current_node` y después se le aplica el apply a este nuevo nodo. Es este método el que nos devuelve un nodo que es insertado en `frontier`, y, como es de esta misma cola de donde se toma el valor de `current_state` antes de volver a empezar el while si es necesario, la información de pertenencia o no de elementos permanece para los nodos hijos constantemente.

(c) Rellena los datos de la tabla con el resultado de aplicar:  
**./practica2 ./mapas/mapa30.map 1 2 3 13 4 4 4 4 13 13**

ScreenShot		
Instantes de simulación no consumidos	2992	
Tiempo Consumido	0.013611	
Nivel Final de Batería	2981	



(d) Rellena los datos de la tabla con el resultado de aplicar:  
**./practica2 mapas/mapa50.map 1 2 19 7 6 3 3 3 16 13**

Screenshot	
Instantes de simulación no consumidos	2982
Tiempo Consumido	1.1352
Nivel Final de Batería	2888

(e) Rellena los datos de la tabla con el resultado de aplicar:

**./practica2 mapas/paldea2.map 1 2 82 49 0 73 52 0 25 25**



<b>Instantes de simulación no consumidos</b>	2926
<b>Tiempo Consumido</b>	5.15327
<b>Nivel Final de Batería</b>	2845

### Nivel 3-A\* para el agente colaborador

- (a) ¿Qué diferencia este algoritmo del de Dijkstra que tuviste que implementar en el nivel anterior? Enumera los cambios y describe brevemente cada uno de ellos y que han implicado en la implementación.

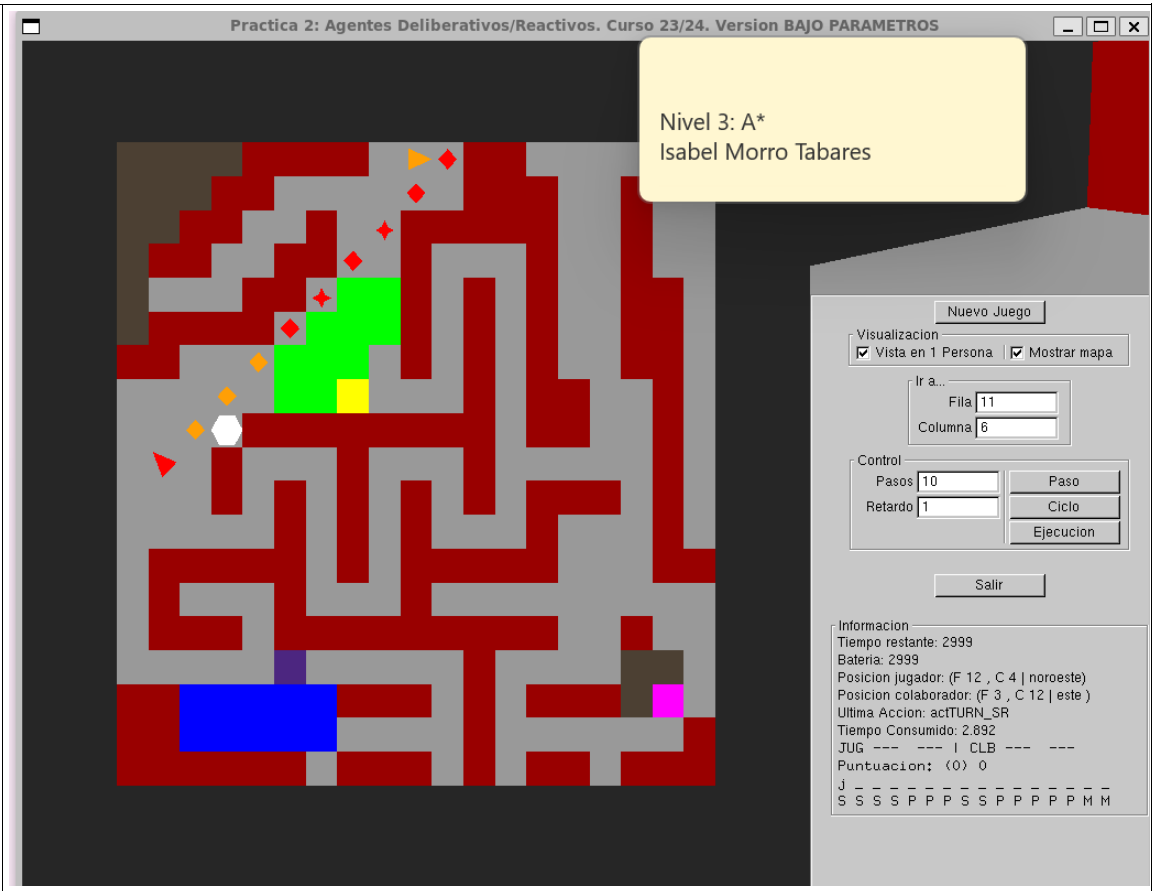
En este nivel el objetivo es llevar al colaborador a la casilla final con el menor coste posible. Para ello se han llevado a cabo una serie de cambios:

- El estado del nivel 3 incluye toda la información mencionada en el nivel 1, además de booleanos que controlan los elementos del jugador y el colaborador por separado. Al igual que en el nivel 2, los operadores  $<$  y  $==$  del estado son los que comparan todos los elementos y posiciones. Ahora bien, en el nodo del nivel 3, se mantiene la variable “costo” de nivel 2 con diferente nombre (“g”), y se añaden las variables “h” y “f”. Es “g” la que guarda el coste del camino ya realizado del inicio del juego al estado actual (lo hace del mismo modo que en el nivel 2), en “h” se guarda la heurística, y “f” es la suma de “h” y “g”. En este nivel, será f quien determine que nodo es menor, y lo será aquel en donde “f” tenga un valor mayor (misma explicación que en el nivel 2).
- En este nivel he añadido métodos necesarios, como el que comprueba la obtención de elementos, ya no solo de jugador, sino de ambos. Además del método CosteEstimado, que es aquel que nos dará el valor de la heurística, de “h”.
- El algoritmo, denominado Aestrella, es una mezcla entre el nivel 1 y 2, por un lado, se mantiene la creación de los nodos hijos de todas las acciones posibles por ambos agentes y las comprobaciones pertinentes del apply para llevar a cabo cada una de las acciones. Por otro lado, como en el nivel 2, se crea la cola de prioridad de nodos y el set de estados, se comprueba la obtención de elementos al comienzo del algoritmo y cada vez que se llama a apply. Es en este método donde, en caso de que las acciones se puedan llevar a cabo, se suma el costo de la acción del jugador y del colaborador, y al final de esta se invoca la llamada a CosteEstimado para obtener el valor de la heurística; pero la estructura de algoritmo y la forma de usarlo es la misma que en el nivel 2. Lo único que cambia es la forma de ordenar la cola de prioridad, que es mediante el valor de “f”.

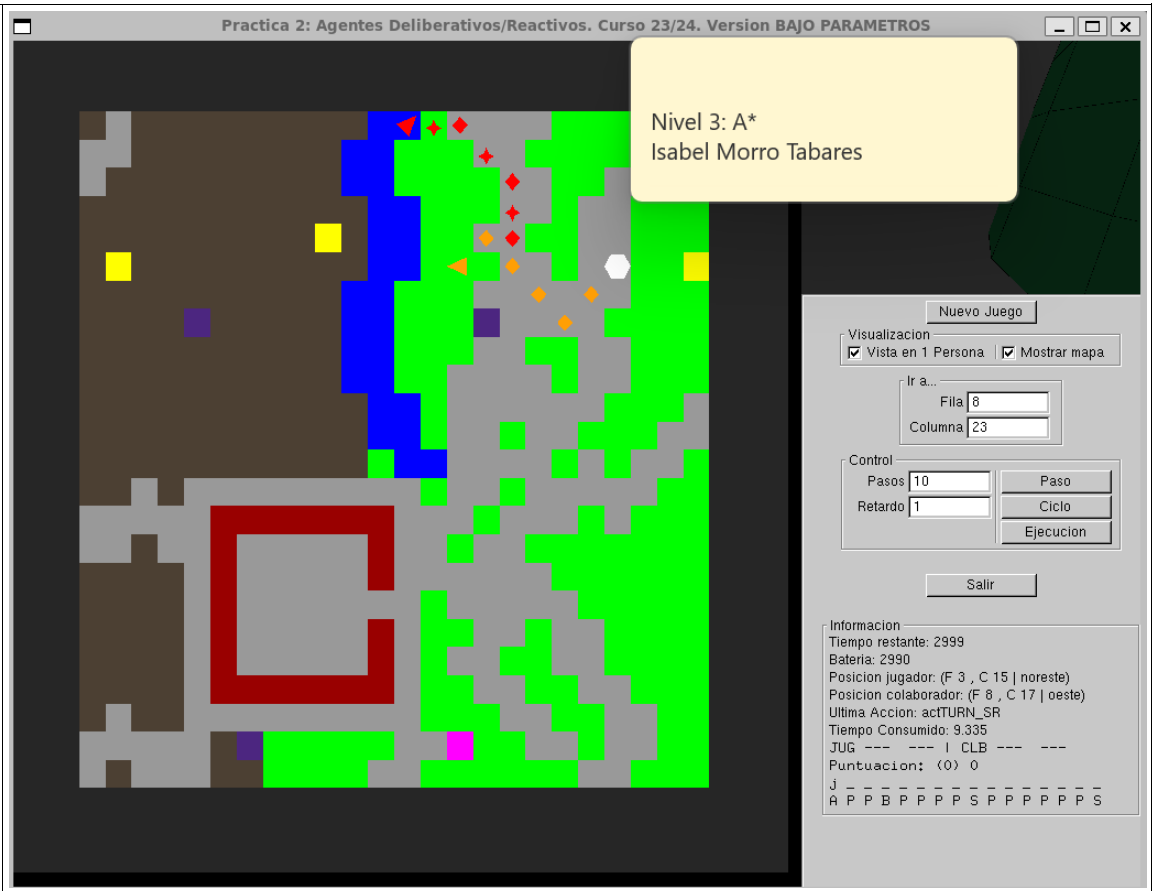
- (b) Describe la heurística utilizada para resolver el problema.

La heurística toma el máximo entre el valor absoluto de la diferencia de la columna donde se encuentra el colaborador en el momento de la llamada y de la columna objetivo, y el valor absoluto de la diferencia de la fila donde se encuentra el colaborador y la fila objetivo, lo que nos proporciona una idea a primera vista de en qué nodos se encuentra más cerca el colaborador del objetivo.

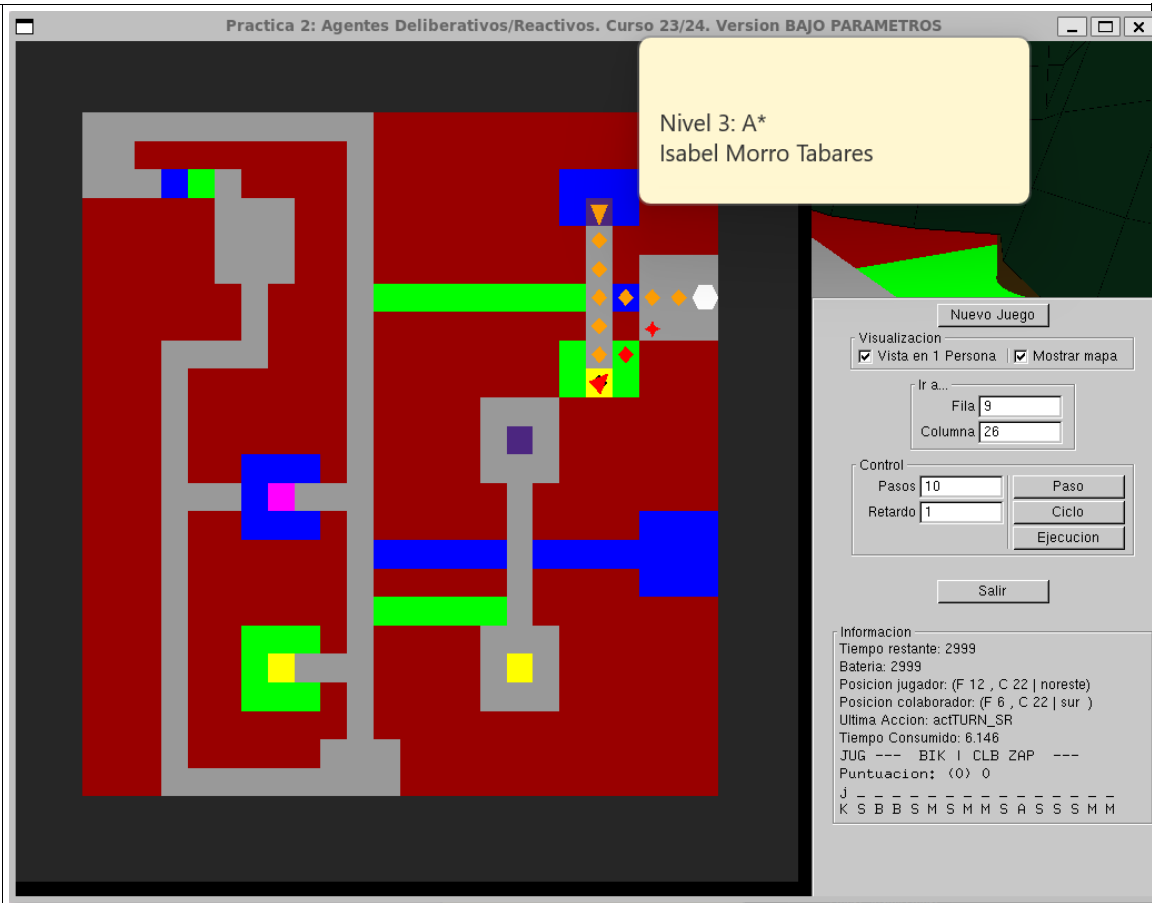
(c) Rellena los datos de la tabla con el resultado de aplicar:  
**./practica2 mapas/lumi25.map 1 3 12 4 6 3 12 2 11 6**

ScreenShot	
Instantes de simulación no consumidos	2968
Tiempo Consumido	2.89267
Nivel Final de Batería	2970

(d) Rellena los datos de la tabla con el resultado de aplicar:  
**./practica2 mapas/mapa30.map 1 3 3 15 0 8 17 6 8 23**

ScreenShot	
<b>Instantes de simulación no consumidos</b>	2967
<b>Tiempo Consumido</b>	9.33548
<b>Nivel Final de Batería</b>	2748

(e) Rellena los datos de la tabla con el resultado de aplicar:  
**./practica2 mapas/scape2.map 1 3 12 22 0 6 22 4 9 26**

ScreenShot	
<b>Instantes de simulación no consumidos</b>	2971
<b>Tiempo Consumido</b>	6.14631
<b>Nivel Final de Batería</b>	2966

## Nivel 4-Reto (Max. Puntuación en misiones)

- (a) Haz una descripción general de la estrategia con la que has abordado este nivel. Indica bajo qué criterios es el jugador o el colaborador el que va al objetivo. Explica brevemente las razones de esos criterios.

En mi implementación, es solo el jugador aquel que va en busca de los objetivos. La estrategia es bastante sencilla, se busca el mínimo camino que lleva al jugador desde su ubicación al objetivo. Lo explico con más detalle en los siguientes apartados.

- (b) ¿Qué algoritmo o algoritmos de búsqueda usas en el nivel 4? Explica brevemente la razón de tu elección.

En el nivel 4 uso el algoritmo de coste uniforme del nivel 2, pues lo más oportuno es que este llegue con el menor gasto de batería posible. He modificado el apply, de forma que tome como válidos aquellos nodos donde, en caso de caminar o correr, el nodo sea transitable si es conocido, o bien si es desconocido.

- (c) ¿Bajo qué condiciones replanifica tu agente?

El jugador replanifica siempre que en su visión encuentre que la casilla a la que avanza, o cruza (en el caso de correr), es transitable (ni muro ni precipicio ni colaborador) o bien si ha habido una colisión. En cuyo caso, se fuerza que la acción siguiente a realizar sea actIDLE, y se formula otro plan.

- (d) Explica el valor de coste que le has dado a la casilla desconocida en la construcción de planes cuando el mapa contiene casillas aún sin conocer. Justifica ese valor.

No le he dado ningún coste, pues buscaba que descubriera el mayor porcentaje de mapa posible al comenzar.

- (e) ¿Has tenido en cuenta la recarga de batería? En caso afirmativo, describe la política usada por tu agente para proceder a recargar.

Sí, pero esta comentada por que me daba fallos, cuando llegaba el momento de ir a por batería se quedaba en bucle en la acción actIDLE. La idea era que, a medida que el jugador va descubriendo el mapa, a la hora de pintar, si alguna casilla de su visión era una casilla de batería, guarda su ubicación, y en caso de tener vida suficiente y niveles de batería bajos, el objetivo ya no era el marcado por la práctica, si no la casilla de batería.

- (f) Añade aquí todos los comentarios que desees sobre el trabajo que has desarrollado sobre este nivel, que consideras son importantes para evaluar el grado en el que te has implicado en la práctica y que no se puede deducir de la contestación a las preguntas anteriores.

Al igual que con la batería, intenté que el jugador tomará elementos de la misma forma, pero tampoco mejoraba los resultados y lo borré.

Anteriormente, tenía otra versión que movía al colaborador de forma que el jugador siempre pudiera ver la casilla que tenía delante, y también intenté que se calculara tanto un camino donde se moviera el colaborador y otro donde lo hiciera el jugador, con los algoritmos del nivel 2 y 3, y se escogía uno u otro tomando como criterio el camino de menor coste y el mayor número de casillas conocidas entre las submatrices que forman la posición del jugador con el objetivo, y el colaborador con el objetivo.

No obstante, no conseguí que estas implementaciones dieran mejores resultados que la entregada, por ello solo muevo al jugador.

- (g) Rellena los datos de la tabla con el resultado de aplicar

**./practica2SG mapas/mapa30.map 1 4 4 4 6 12 12 2 8 8 16 6 26 9 25 16 3 3  
5 10 16 14 18 10 21 4 7 4 24 4 12 3 6 15 25 18 8 19 15 14 23 15 26 8 10 24 19  
26 25 7 16 11 22 15 20 15 22 19 10 20 4 13 26 24 9 6 26 10 17 19 25 13 24 20  
26 19 12 18 8 23 9 13 6 5 8 16 12 5 3 14 11 22 11 8 6 17 7 4 21 3 23 4 15 5 7  
23 21 19 4 15 6 13 24 17 6 26 5 4 24 10 16 17 13 20 22 9 26 22 22 24 14 4 24  
26 7 18 6 21 9 9 18 6 19 15 16 21 3 14 13 10 25 13 17 24 7 20 14 14 9 21 5 18  
20 20 19 10 5 18 18 21 9 22 20 19 7 15 26 20 10 17 17 19 8 23 8 9 9 5 20 3 20  
11 6 23 16 8 26 14 17 4 8 25 14 13 25 14 8 5 20 21 4 18 14 25**

<b>Instantes de simulación no consumidos</b>	2009
<b>Tiempo Consumido</b>	3.41641
<b>Nivel Final de Batería</b>	0
<b>Objetivos Alcanzados</b>	75
<b>Puntuación</b>	75



- (h) Rellena los datos de la tabla con el resultado de aplicar **./practica2SG mapas/mapa75.map**
- 1 4 11 47 0 6 47 4 9 51 19 33 29 31 33 31  
54 43 51 67 59 68 57 29 47 16 68 6 29 4 3 32 14 25 57 20 41 35 47 27 38 71  
68 47 53 10 18 10 60 11 16 38 62 45 22 47 39 37 57 39 57 30 66 21 26 60 68 5  
6 14 49 66 26 4 4 17 63 19 40 28 57 38 50 39 69 69 54 21 55 47 26 51 12 33  
66 62 24 17 67 26 51 71 44 49 64 32 69 29

<b>Instantes de simulación no consumidos</b>	2599
<b>Tiempo Consumido</b>	2.64014
<b>Nivel Final de Batería</b>	0
<b>Objetivos Alcanzados</b>	12
<b>Puntuación</b>	12

- (i) Rellena los datos de la tabla con el resultado de aplicar **./practica2SG mapas/pinkworld.map**
- 1 4 46 26 2 41 27 2 44 46 26 59 26 10  
59 10 59 63 70 38 41 35 47 27 38 71 68 47 18 10 60 11 30 9 66 21 68 5 71 39  
6 14 49 66 4 17 40 28 50 39 69 69 12 33 66 62 67 26 51 71 44 49 64 32 69 29  
68 36

<b>Instantes de simulación no consumidos</b>	2483
<b>Tiempo Consumido</b>	26.554
<b>Nivel Final de Batería</b>	0
<b>Objetivos Alcanzados</b>	7
<b>Puntuación</b>	7