UNIVERSIDAD INTERNACIONAL DE LA RIOJA

MÁSTER UNIVERSITARIO EN INTELIGENCIA ARTIFICIAL

RESOLUCIÓN DE PROBLEMA MEDIANTE BÚSQUEDA HEURÍSTICA

ACTIVIDAD 1: EL AMAZON ROBOT

POR

ANDRÉS SÁNCHEZ SÁNCHEZ

06, MAYO DE 2021

ÍNDICE

IMPLEMENTACIÓN DEL ALGORITMO A*

0 DESARROLLO DE ACTIVIDAD

Se tiene que implementar un algoritmo de A* cuyas funciones consistirán en que un robot se desplace desde una posición inicial (meta) hasta donde se encuentra un repositorio (destino), cargar este repositorio y llevarlo a unas posiciones predeterminadas.

En el ejercicio que se plantea se tienen tres repositorios y el robot tendrá que trasportarlos desde donde se encuentran a los puntos que se indican. Para ello las consideraciones que se hacen serán las siguiente:

Para ello las consideraciones que se hacen serán las siguiente teniendo en cuenta que hay que hacer tres viajes.

1. Hay que decidir cuál será el primer viaje, el segundo y el tercero, para ello se calculará la h* de ida y vuelta y se empezará por la que menor coste tenga, después se recalculará para ver cuál de las otras dos tiene menor h* desde la nueva posición del robot para decidir qué acción se ejecuta en segundo lugar.

2. Se considera que los repositorios no se pueden atravesar, es decir que los repositorios que no se muevan son considerados obstáculos cuando el robot esté transportando otro, y si se considera que el robot cuando se desplaza solo, puede pasar por debajo de los repositorios.

3. Habrá dos mapas, el original donde se reflejarán todo en cadena de caracteres y otro numérico para poder hacer mejor todas las consideraciones pertinentes.

A continuación, se muestra la disposición del mapa, indicando donde está el repositorio 1 (M1), el repositorio 2 (M2), el repositorio 3 (M3) y el robot (R). Las almohadillas (#) son considerados obstáculos que el robot no puede atravesar.

En el mapa numérico se traducirá todo esto como, M1=5, M2=4, M3=3, R=2, #=1 y los espacios vacíos que no son obstáculos como un 0.

A continuación, se muestra tanto el mapa original como el mapa objetivo:

M1	#		M3
	#		
M2		R	

	#		
	#		
	M3	M2	M1

1 DISEÑO DE FUNCIONES

Una vez determinado el punto de partida y el punto de meta, hay que definir las funciones que se usarán para llevar a cabo este cometido.

Inicialmente se creará una unción llamada *convermap*, donde se le pará el parámetro del mapa con notación, o bien de carácter, o bien numérico. Esta función tiene por objetivo traducir el mapa a numérico

para poder manipularlo y a carácter para que sea legible. El mapa cuando se convierte a código numérico quedará tal que así:

5	1	0	3
	1	0	0
4	0	2	0
0	0	0	0

Se explica a continuación con un pseudocódigo de cómo funciona esta función.

```
// Pasar datos mapa
    //Si mapa es str
        //Recorrer matriz
            //Asignar número correspondiente a su equivalencia de letras
    //Si mapa es número
        //Recorrer matriz
            //Asignar letra correspondiente a su equivalencia de números
    //Devolver mapa
```

Después de traducir toda la información para poder manipularla, se tiene que automatizar el proceso para que se decida cual es el orden del trabajo de transporte de los repositorios hacia su destino. Esto se ha realizado con una función llamada *h_ida_vuelta*, a la que se le pasa por parámetro la posición del robot, y las posiciones de los repositorios. Para hacer esto, la función calcula la de menor h estimada, para que el robot empiece por ese repositorio, y después, desde la meta de ese repositorio, se vuelve a calcular la h estimada para decidir cuál sería el siguiente repositorio a transportar.

En el caso de que hubiera dos h estimadas iguales, la propia configuración del código determina el orden de ejecución del robot.

Después de haber pasado por esta función, ya no importará si es repositorio M1, M2 o M3 y estas variables serán sustituidas por seis variables (la ida y la vuelta del robot) en las que se indica desde el viaje 1 hasta el viaje 6.

Se explica a continuación con un pseudocódigo de cómo funciona esta función.

```
//Pasar coordenadas robot y repositorios
    //Calcular h en ida y vuelta
    //Asignar h_primera la h mínima
    // Si h_primera es igual h1
        //Asignar m_primera a la meta M1
        //Calcular h nuevas desde meta M1
        //Asignar h_segunda el valor mínimo de una de las h nueva y h_tercera al contrario
```

```
//Repetir proceso anterior 2 veces más para opciones h1 y h2

//Devolver h_primera, h_segunda, h_tercera, m_primera, m_segunda, m_tercera
```

Una vez se ha implementado esta función, se implementará la clase *Nodo*, que es donde se almacenan todos los datos del nodo con sus funciones de costes, la función heurística y la función de evaluación, además de si ese nodo tiene padre.

También, la clase nodo, además del método mágico *__init__*, tiene el método mágico *__eq__* que sirve para comprobar si un dato es igual a otro, y se hace pensando en que se compruebe que si las nuevas coordenadas que han sido creadas no sean iguales y entre en un bucle infinito.

Cabe destacar que la función de evaluación es la que aparece en la siguiente ecuación:

$$f(n) = g(n) + h^*(n)$$

Siendo f(n) la función de evaluación, g(n) la función de coste de ir desde el nodo inicial al nodo n y h*(n) se corresponde con la función heurística que mide la distancia estimada desde n a algún nodo meta.

2 IMPLEMENTACIÓN DE A*

La función más importante para el desarrollo de la actividad, es la implementación del algoritmo A* que se hará con la función *a_estrella*, a la cual, se le pasará el mapa, las coordenadas de partida y las coordenadas de llegada. En ella se calcula el mejor desplazamiento que debería seguir el robot siguiendo la distancia Manhattan. Hay que tener en cuenta que a veces los repositorios son obstáculos y otras veces no.

A continuación, se explica paso lo que se pretende con un pseudocódigo.

```
//Pasar mapa, coordenadas de inicio y coordenadas de fin

    //Inicializar nodo_salida con clase Nodo

    //Inicializar nodo_llegada con clase Nodo

    //Inicializar lista_abierta y lista_cerrada

    //Incluir nodo_salida en lista abierta

    //Repetir bucle mientras lista_abieta>0

        //Inicializar el nodo actual con el valor inicial de lista_abierta

        //Recorrer lista_abierta

            //Si f.eval de valor de lista_abierta < f.eval actual

                //Asignar nodo actual el valor al que apunta lista_abierta

        //Sacar valor actual de lista_abierta

        //Incluir actual en lista_cerrada

        //Si actual == nodo_llegada

            //Inicializar ruta
```

```
            //Asignar padre de actual
            //Bucle de padre ¡= None
                  //Añadir coordenadas de padre a ruta
                  //Asignar padre el valor de su padre
            //Devolver ruta
      //Inicializar hijos
      //Inicializar en variable movimientos los movimientos permitidos
      //Bucle para recorrer nodos vecinos
            //Asignar nodo_evaluado con datos del nodo vecino
            //Si nodo_evaluado se sale del mapa
                  //Saltar al siguiente
            //Si nodo_evaluado está en un obstáculo
                  //Saltar al siguiente
            //Si condiciones anteriores no se cumplen
                  //Asignar nuevo_nodo clase Nodo indicando el padre y coordenadas
            //Recorrer lista hijos
                  //Si hay hijos en lista_cerrada
                        //Saltar al siguiente
                  //Asignar f.eval, h y g.cost al hijo que no está en lista_cerrada
                  //Recorrer lista_abierta
                        //Si hijo está en lista_abierta & h.g_n>hijo.g_n de lista_abierta
                              //Saltar al siguente
                  //Incluir hijo en lista_abierta
```

El código de esta función devuelve una lista de coordenadas que indican el camino que debe tomar el robot atendiendo a las restricciones que se presentan.

3 EJECUCIÓN DEL ROBOT

Una vez se hayan implementado todas las funciones para desarrollar la actividad del robot de Amazon hay se ejecuta la función *h_ida_vuelta* pasándole la posición del robot (R=[2,2) y la posición de los repositorios (posM1=[0,0], posM2=[2,0] y posM3[0,3]) y estos datos se estandarizan en seis variables que representan las coordenadas de ida y vuelta de los tres viajes que tiene que hacer el robot. Las variables son ida1, ida2, ida3, vuelta1, vuelta2, vuelta3.

Esta función decide que el primer viaje debe ser el del repositorio M2 y como se indicó en el principio, a la ida no habría obstáculos y a la vuelta, los repositorios son obstáculos. A continuación se representa el mapa numérico de que como hace el camino de ida y vuelta después de ejecutar la función *a_estrella(mapa,R,ida1)* y *a_estrella(mapa, ida1,vuelta1)* actualizando el mapa entremedia con las coordenadas extraídas de la función *h_ida_vuelta*.

0	1	0	0
0	1	0	0
0	0	R	0
0	0	0	0

1	1	0	1
0	1	0	0
R	0	0	0
0	0	0	0

La ruta obtenida es [[2, 2], [2, 1], [2, 0], [2, 1], [2, 2], [3, 2]] expresado por coordenadas t teniendo un coste de función de evaluación de 8.

Para el segundo viaje la función de ida y vuelta para el segundo viaje será *a_estrella(mapa,vuelta1,ida2)* y *a_estrella(mapa, ida2,vuelta2)* y en esta ocasión tendremos la ruta tal y como se muestra:

0	1	0	0
0	1	0	0
0	0	0	0
0	0	R	0

1	1	0	R
0	1	0	0
0	0	0	0
0	0	1	0

La ruta obtenida es [[3, 2], [3, 3], [2, 3], [1, 3], [0, 3], [0, 2], [1, 2], [2, 2], [2, 1], [3, 1]] expresado por coordenadas t teniendo un coste de función de evaluación de 12.

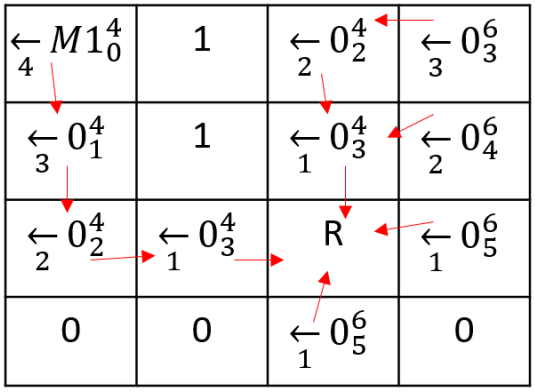
Y por último, el viaje final tiene las condiciones de *a_estrella(mapa,vuelta2,ida3)* y *a_estrella(mapa, ida3,vuelta3)*:

0	1	0	0
0	1	0	0
0	0	0	0
0	R	0	0

R	1	0	0
0	1	0	0
0	0	0	0
0	1	1	0

La ruta obtenida es [[3, 1], [3, 0], [2, 0], [1, 0], [0, 0], [1, 0], [2, 0], [2, 1], [2, 2], [2, 3], [3, 3]] expresado por coordenadas t teniendo un coste de función de evaluación de 12.

Para observar el procedimiento que sucede para elegir el camino, a continuación, se muestra una imagen donde se ve el recorrido desde las coordenadas [2,2] a [0,0] estando en la parte superior la función de evaluación, en la inferior derecha la h estimada y en la derecha el coste. Además, las flechas rojas apuntan al padre del nodo.



4 CÓDIGO DE CUADERNO DE JUPYTER NOTEBOOK

A continuación, se mostrará el código del cuaderno de Jupyter Notebook con sus ejecuciones, y además podrá ser visionado en con todas las anotaciones y explicaciones pertinentes en cada celda del cuaderno para así, tener una mayor comprensión del mismo.

Actividad de Laboratorio de Razonamiento y planificación Automática ¶

Se tienen que hacer tres viajes, por tanto hay que tener una serie de consideraciones previamente.

1. Hay que decidir cual será el primer viaje, el segundo y el tercero, para ello se calculará la h* de ida y vuelta y se empezará por la que menor coste tenga, después se recalculará para ver cual de las otras dos tiene menor h* desde la nueva posición del robot para decidir que acción se ejecuta en segundo lugar.
2. Se considera que los repositorios no se pueden atravesar, es decir que los repositorios que no se muevan son considerados obstáculos cuando el robot esté transportándolos, y si se considera que el robot cuando se desplaza solo, puede pasar por debajo de los repositorios.
3. Habrá dos mapas, el original donde se reflejarán todo en cadena de caracteres y otro numérico para poder hacer mejor todas las consideraciones pertinentes.

A continuación, se muestra la disposición del mapa, indicando donde está el repositorio 1 (M1), el repositorio 2 (M2), el repositorio 3 (M3) y el robot (R). Las almohadillas (#) son considerados obstáculos que el robot no puede atravesar. En el mapa numérico se traducirá todo esto como:

1. M1=5
2. M2=4
3. M3=3
4. R=2
5. #=1
6. Los espacios vacíos que no son obstáculos como un 0

Para ello se creará una función para traducir de uno a otro y viceversa

In [1]:

```
#Se representa en una matriz 4x4
mapa=[['M1','#',' ','M3'],[' ','#',' ',' '],['M2',' ','R',' '],[' ',' ',' ',' ']]
mapa
```

Out[1]:

```
[['M1', '#', ' ', 'M3'],
 [' ', '#', ' ', ' '],
 ['M2', ' ', 'R', ' '],
 [' ', ' ', ' ', ' ']]
```

La disposición final que se quiere es la que se mustra a continuación

In [2]:

```
#Se representa en una matriz 4x4
meta=[[' ','#',' ',' '],[' ','#',' ',' '],[' ',' ',' ',' '],[' ','M3','M2','M1']]
meta
```

Out[2]:

```
[[' ', '#', ' ', ' '],
 [' ', '#', ' ', ' '],
 [' ', ' ', ' ', ' '],
 [' ', 'M3', 'M2', 'M1']]
```

In [3]:

```
import numpy as np
def convermap(mapa):
    if mapa[0][0]=='M1' or mapa[0][0]=='M2' or mapa[0][0]=='M3' or\
        mapa[0][0]=='R' or mapa[0][0]==' ':

        for i in range(0,len(mapa)):
            for j in range(0,len(mapa[len(mapa)-1])):
                if mapa[i][j]=='M1':
                    mapa[i][j]=5
                elif mapa[i][j]=='M2':
                    mapa[i][j]=4
                elif mapa[i][j]=='M3':
                    mapa[i][j]=3
                elif mapa[i][j]=='R':
                    mapa[i][j]=2
                elif mapa[i][j]=='#':
                    mapa[i][j]=1
                elif mapa[i][j]==' ':
                    mapa[i][j]=0
        return mapa
    else:
        for i in range(0,len(mapa)):
            for j in range(0,len(mapa[len(mapa)-1])):
                if int(mapa[i][j])==0:
                    mapa[i][j]=' '
                elif int(mapa[i][j])==1:
                    mapa[i][j]='#'
                elif int(mapa[i][j])==2:
                    mapa[i][j]='R'
                elif int(mapa[i][j])==3:
                    mapa[i][j]='M3'
                elif int(mapa[i][j])==4:
                    mapa[i][j]='M2'
                elif int(mapa[i][j])==5:
                    mapa[i][j]='M1'
                elif int(mapa[i][j])==6:
                    mapa[i][j]='M3'
                elif int(mapa[i][j])==8:
                    mapa[i][j]='M2'
                elif int(mapa[i][j])==10:
                    mapa[i][j]='M1'
        return mapa
```

In [4]:

```
x=convermap(mapa)
x
```

Out[4]:

```
[[5, 1, 0, 3], [0, 1, 0, 0], [4, 0, 2, 0], [0, 0, 0, 0]]
```

In [5]:

```
y=convermap(x)
y
```

Out[5]:

```
[['M1', '#', ' ', 'M3'],
 [' ', '#', ' ', ' '],
 ['M2', ' ', 'R', ' '],
 [' ', ' ', ' ', ' ']]
```

Ahora habría que hacer otra función que permitira calcular la h* de ida y vuelta para determinar cual sería el primer repositorio a mover, cual sería el segundo y cual sería el tercero. para ello hay que indicar donde está el situado el roboto, donde está cada repositorio y cual es su meta.

In [6]:

```
def h_ida_vuelta(posR,posM1,posM2,posM3):
    #Se calculará esta eurística con la distancia Manhatttan
    #Es conocido como premisa de la actividad que las metas de cada repositorio están p
redefinidas
    #Meta M1=[3,3]
    #Meta M2=[3,2]
    #Meta M2=[3,1]
    h1=sum([abs(posR[0]-posM1[0])+abs(posM1[0]-3),abs(posR[1]-posM1[1])+abs(posM1[1]-3
)])
    h2=sum([abs(posR[0]-posM2[0])+abs(posM2[0]-3),abs(posR[1]-posM2[1])+abs(posM2[1]-2
)])
    h3=sum([abs(posR[0]-posM3[0])+abs(posM3[0]-3),abs(posR[1]-posM3[1])+abs(posM3[1]-1
)])
    h_primera=min(h1,h2,h3)
    #El siguente camino para averiguar cual tiene menor coste se hace desde una de las
 metas
    if h_primera==h1:
        h2=sum([abs(3-posM2[0])+abs(posM2[0]-3),abs(3-posM2[1])+abs(posM2[0]-2)])
        h3=sum([abs(3-posM3[0])+abs(posM3[0]-3),abs(3-posM3[1])+abs(posM3[0]-1)])
        h_segunda=min(h2,h3)
        h_tercera=max(h2,h3)
        h_primera=posM1
        m_primera=[3,3]
        if h_segunda==h2:
            h_segunda=posM2
            h_tercera=posM3
            m_segunda=[3,2]
            m_tercera=[3,1]
        else:
            h_segunda=posM3
            h_tercera=posM2
            m_segunda=[3,1]
            m_tercera=[3,2]
    elif h_primera==h2:
        h1=sum([abs(3-posM1[0])+abs(posM1[0]-3),abs(2-posM1[1])+abs(posM1[1]-3)])
        h3=sum([abs(3-posM3[0])+abs(posM3[0]-3),abs(2-posM3[1])+abs(posM3[1]-1)])
        h_segunda=min(h1,h3)
        h_tercera=max(h1,h3)
        h_primera=posM2
        m_primera=[3,2]
        if h_segunda==h1:
            h_segunda=posM1
            h_tercera=posM3
            m_segunda=[3,3]
            m_tercera=[3,1]
        else:
            h_segunda=posM3
            h_tercera=posM1
            m_segunda=[3,1]
            m_tercera=[3,3]
    elif h_primera==h3:
        h1=sum([abs(3-posM1[0])+abs(posM1[0]-3),abs(1-posM1[1])+abs(posM1[1]-3)])
        h2=sum([abs(3-posM2[0])+abs(posM2[0]-3),abs(1-posM2[1])+abs(posM2[1]-2)])
        h_segunda=min(h1,h2)
        h_tercera=max(h1,h2)
        h_primera=posM3
        m_primera=[3,1]
        if h_segunda==h2:
            h_segunda=posM2
```

```
            h_tercera=posM1
            m_segunda=[3,2]
            m_tercera=[3,3]
        else:
            h_segunda=posM1
            h_tercera=posM2
            m_segunda=[3,3]
            m_tercera=[3,2]
    return h_primera, h_segunda, h_tercera, m_primera, m_segunda, m_tercera
```

In [7]:

```
#Aquí se ejemplifica como funciona la función h_ida_vuelta
travel1, travel2, travel3, meta1, meta2, meta3 = h_ida_vuelta([2,2],[0,0],[2,0],[0,3])
print(f'El primer viaje será el de la coordenada {travel1}\nEl segundo viaje será el de
la coordenada{travel2} \
        \nEl tercer viaje será el de las coordenadas {travel3}')
print(f'La meta 1 será el de la coordenada {meta1}\nLa meta 2 será el de la coordenada
{meta2} \
        \nLa meta 3 viaje será el de las coordenadas {meta3}')
```

```
El primer viaje será el de la coordenada [2, 0]
El segundo viaje será el de la coordenada[0, 3]
El tercer viaje será el de las coordenadas [0, 0]
La meta 1 será el de la coordenada [3, 2]
La meta 2 será el de la coordenada[3, 1]
La meta 3 viaje será el de las coordenadas [3, 3]
```

Hay que decir, que los costes de carga y descarga no se han contabilizado anteriormente ya que ese incremento de coste afecta a todos los repositorios por igual. Una vez creada la función que define cual sería el orden de actuación para buscar los repositorios, se implementa el código y para ello se creará la clase nodo, en la cual se podrán diferenciar los distintos nodos con su función de coste actualizada y donde te indica quien es el padre de ese nodo y su posición

In [8]:

```
class Nodo():
    #Se crea la función init para inicializar los nodos.
    def __init__(self,padre=None, coordenada=None):
        self.padre=padre
        self.coordenada=coordenada
        self.g_n=0
        self.h_n=0
        self.f_n=0
        #Esta función es un método mágico que sirve para comprobar si un dato es igual
 a otro
        #Esta función en este caso está pensada para que compruebe si la nueva coordena
da es igual
        #a alguna anterior para sí no entrar en un bucle infinito y que luego evalúe co
nstantemente
        #coordenadas ya evaluadas
    def __eq__(self,other):
        return self.coordenada==other.coordenada
```

En la función que se mostrará a continuación, se implementará el algoritmo de A*

In [9]:

```
#Esta función, devolverá los valores pasados por coordenadas
def a_estrella(mapa,inicio,objetivo):
    mapa=np.array(mapa)
    #Primero se crea el nodo de inicio y y final,inicializandolos sin padres y sin valo
r en su función de coste
    nodo_salida=Nodo(padre=None,coordenada=inicio)
    nodo_salida.g_n=0
    nodo_salida.h_n=0
    nodo_salida.f_n=0

    nodo_llegada=Nodo(padre=None,coordenada=objetivo)
    nodo_llegada.g_n=0
    nodo_llegada.h_n=0
    nodo_llegada.f_n=0

    #Después de inicializar los nodos, se tiene que crear la lista abierta y la lista c
errada tal como
    #se especifica en el algoritmo A*
    lista_abierta=[]
    lista_cerrada=[]

    #Sabemos que tenemos que introducir el nodo de inicio en la lista abierta antes de
 expandirlo
    lista_abierta.append(nodo_salida)
    #En el siguiente bucle será donde se encuentr el camino que se quiere de A* y termi
nará cuando ya no haya
    #datos en la lista abierta
    while len(lista_abierta)>0:
        #En el primer valor de la lista abierta estará el nodo actual con el que se est
á trabajando
        actual=lista_abierta[0]
        for i in range(0,len(lista_abierta)):
            if lista_abierta[i].f_n<actual.f_n:
                actual=lista_abierta[i]
        #Hay que sacarlo de la lista abierta e incluirlo en la lista cerrada
        lista_abierta.pop(lista_abierta.index(actual))
        lista_cerrada.append(actual)

        #Si encuentra la meta se terminará el bucle
        if actual==nodo_llegada:
            ruta=[]
            padre=actual
            while padre is not None:
                ruta.append(padre.coordenada)
                padre=padre.padre
            #Devuelve la ruta deseada A*
            return list(reversed(ruta))
        hijos=[]
        #Este array se usa paraincrementar o disminuir en una unidad la posición del no
do a futuro
        movimiento=[[0,-1],[0,1],[-1,0],[1,0]]
        for nuevo_movimiento in movimiento:
            nodo_evaluado=[int(actual.coordenada[0])+int(nuevo_movimiento[0]),
                           int(actual.coordenada[1])+int(nuevo_movimiento[1])]

            #Ahora se añade una condición para evitar que se escoja un camino vetado pa
ra encontrar la ruta
            #Hay que considerar si se trata del viaje de ida o de vuelta
            if nodo_evaluado[0]>(mapa.shape[0]-1) or nodo_evaluado[0]<0 or nodo_evaluad
```

```
o[1]>(mapa.shape[1]-1) \
                    or nodo_evaluado[1]<0:
                #aquí se controla que no se salga del mapa
                pass
            elif int(mapa[nodo_evaluado[0],nodo_evaluado[1]])!=0:
                #aquí se conmprueba que no se transpase con ningún muro
                pass
            else:
                #Si el nodo a desplazarse es válido, se ejecuta este código
                nuevo_nodo=Nodo(padre=actual,coordenada=nodo_evaluado)
                #Se añade a la lista de hijo
                hijos.append(nuevo_nodo)

            #En el siguiente bucle se comprobará quien puede ser hijo de que padre espe
cífico

            for hijo in hijos:
                #Se quiere saber que hijo está en la lista cerrada, significa que no ha
y que evaluarlo y
                #pasará al siguiente para no entrar en un bucle infinito
                for hijo_cerrado in lista_cerrada:
                    if hijo==hijo_cerrado:
                        continue
                #Aqui se evaluará la función de coste de cada coordenada
                hijo.g_n=actual.g_n+1
                hijo.h_n=sum([abs(hijo.coordenada[0]-objetivo[0]),abs(hijo.coordenada[1
]-objetivo[1])])
                hijo.f_n=hijo.g_n+hijo.h_n

                #Si el hijo ya estuviera en la lista abierta y el desplazamiento fuera
 más grande de la
                #unidad se controla con esta parte del código
                for abrir in lista_abierta:
                    if hijo == abrir and hijo.g_n>abrir.g_n:
                        continue
                #Se incluye al hijo en la lista abierta para tener conciencia de que fu
e evaluado con sus datos actualizados
                lista_abierta.append(hijo)
```

Ahora se va a resolver el problema planteado usando las funciones creadas para ello y para ello se mostrará la posición inicial y cual es el objetivo que se pretende alcanzar

In [10]:

```
mapa
```

Out[10]:

```
[['M1', '#', ' ', 'M3'],
 [' ', '#', ' ', ' '],
 ['M2', ' ', 'R', ' '],
 [' ', ' ', ' ', ' ']]
```

In [11]:

```
meta
```

Out[11]:

```
[[' ', '#', ' ', ' '],
 [' ', '#', ' ', ' '],
 [' ', ' ', ' ', ' '],
 [' ', 'M3', 'M2', 'M1']]
```

In [12]:

```
#Para poder trabajar mejor, se convertirá el mapa a código numérico pasándolo al mapa p
rimo
mapa_p=convermap(mapa)
mapa_p
```

Out[12]:

```
[[5, 1, 0, 3], [0, 1, 0, 0], [4, 0, 2, 0], [0, 0, 0, 0]]
```

In [13]:

```
#Una vez convertido a numérico, se tiene que decidir cual será la mejor opción para que
el robot actúe
#Las posiciones son las siguientes
R=[2,2]
posM1=[0,0]
posM2=[2,0]
posM3=[0,3]
#Con esta función determinaremos el orden para que esté automátizado
ida1, ida2, ida3, vuelta1, vuelta2, vuelta3=h_ida_vuelta(R,posM1,posM2,posM3)
```

In [14]:

```
#Hay que eliminar todo el ruido del mapa conviertiéndolo a 0 y 1
for i in range(0,len(mapa_p)):
    for j in range(0,len(mapa_p[len(mapa_p)-1])):
        if mapa_p[i][j]==5 or mapa_p[i][j]==5 or mapa_p[i][j]==4 or mapa_p[i][j]==3 or
mapa_p[i][j]==2:
            mapa_p[i][j]=0

viaje1I=a_estrella(mapa,R,ida1)
#En este punto, se ha cargado uno de los repositorios, así que los otros repositorios p
asan a ser obstáculos
mapa_p[ida2[0]][ida2[1]]=1
mapa_p[ida3[0]][ida3[1]]=1
viaje1V=a_estrella(mapa,ida1,vuelta1)

#A partir de aquí, se convina las dos rutas
ruta_primera=viaje1I
ruta_primera.pop()
ruta_primera.extend(viaje1V)
print(ruta_primera)
```

```
[[2, 2], [2, 1], [2, 0], [2, 1], [2, 2], [3, 2]]
```

Una vez realizado el primer viaje los repositorios cambian de ser obstáculos a poder ser transpasado bajo ellos y cambian las dimensiones del mapa

In [15]:

```
mapa_p[vuelta1[0]][vuelta1[1]]=0
mapa_p[ida2[0]][ida2[1]]=0
mapa_p[ida3[0]][ida3[1]]=0

viaje2I=a_estrella(mapa_p,vuelta1,ida2)
#En este punto, se ha cargado uno de los repositorios, así que los otros repositorios p
asan a ser obstáculos
mapa_p[vuelta1[0]][vuelta1[1]]=1
mapa_p[ida3[0]][ida3[1]]=1
viaje2V=a_estrella(mapa,ida2,vuelta2)

#A partir de aquí, se convina las dos rutas
ruta_segunda=viaje2I
ruta_segunda.pop()
ruta_segunda.extend(viaje2V)
print(ruta_segunda)
```

```
[[3, 2], [3, 3], [2, 3], [1, 3], [0, 3], [0, 2], [1, 2], [2, 2], [2, 1],
[3, 1]]
```

Otra vez se eliminan los obstáculos excepto los muros y se hace el último viaje

In [16]:

```
mapa_p[vuelta1[0]][vuelta1[1]]=0
mapa_p[vuelta2[0]][vuelta2[1]]=0
mapa_p[ida3[0]][ida3[1]]=0

viaje3I=a_estrella(mapa_p,vuelta2,ida3)
#En este punto, se ha cargado uno de los repositorios, así que los otros repositorios p
asan a ser obstáculos
mapa_p[vuelta1[0]][vuelta1[1]]=1
mapa_p[vuelta2[0]][vuelta2[1]]=1
viaje3V=a_estrella(mapa,ida3,vuelta3)

#A partir de aquí, se convina las dos rutas
ruta_tercera=viaje3I
ruta_tercera.pop()
ruta_tercera.extend(viaje3V)
print(ruta_tercera)
```

```
[[3, 1], [3, 0], [2, 0], [1, 0], [0, 0], [1, 0], [2, 0], [2, 1], [2, 2],
[2, 3], [3, 3]]
```