PRÁCTICA1 IA

ÍNDICE

Javier Castro Magro

Raúl Somavilla González

1. INTRODUCCIÓN

La elaboración de esta práctica, la hemos dividido en dos partes fundamentales.

La primera sería el planteamiento de la práctica, así como su comprensión.

La segunda parte consiste en el desarrollo mismo de la práctica.

2. PLANTEAMIENTO DE LA PRÁCTICA

En esta parte, lo que se ha realizado es un análisis de lo que se nos pide y se han tomado las decisiones de por dónde afrontar la práctica.

Empezamos realizando el problema con papel y boli.

En papel planteamos una búsqueda **A*** con la heurística **distancia Manhattan** a la meta.

En caso de que dos nodos tuviesen la misma distancia a la meta, priorizaríamos el **orden de las agujas del reloj**, es decir arriba, derecha, abajo y por último izquierda.

Además, para que el desarrollo del problema en papel **evitamos estados repetidos**.

En este mapa, vemos que el personaje se teletransporta, por lo que en el código lidiaremos con ello como veremos más adelante.

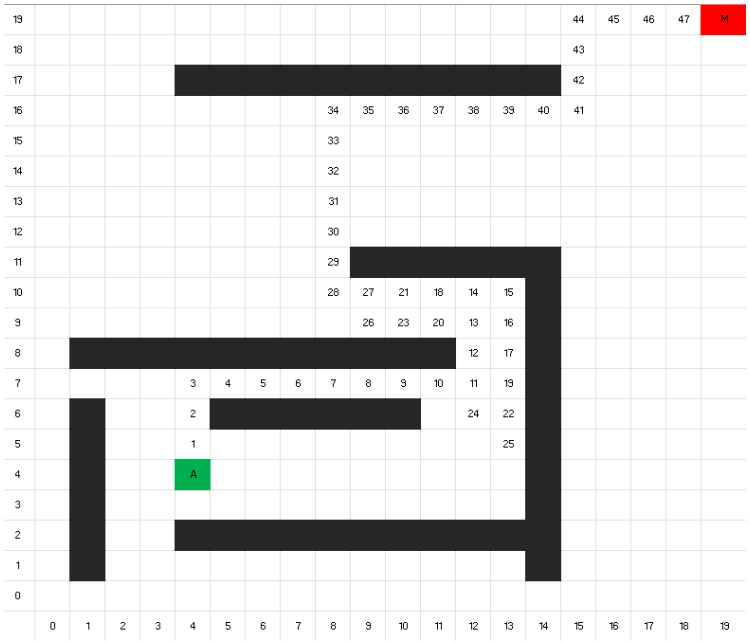
A continuación, se va a mostrar este proceso.

2.1 MAPAS

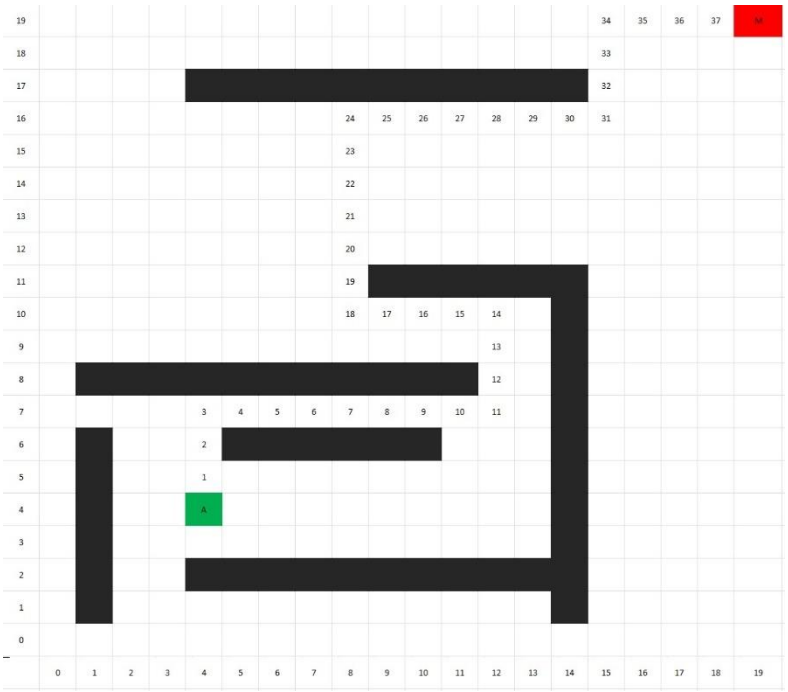
Este primer mapa es el del primer análisis que realizamos.

El path por el que tiene que pasar el agente es el mostrado en el mapa 2. Este camino se genera con la lista de los padres que va desde la meta al origen y luego con el reverse le das la vuelta, para que el agente siga ese camino.

2.1.1 MAPA 1

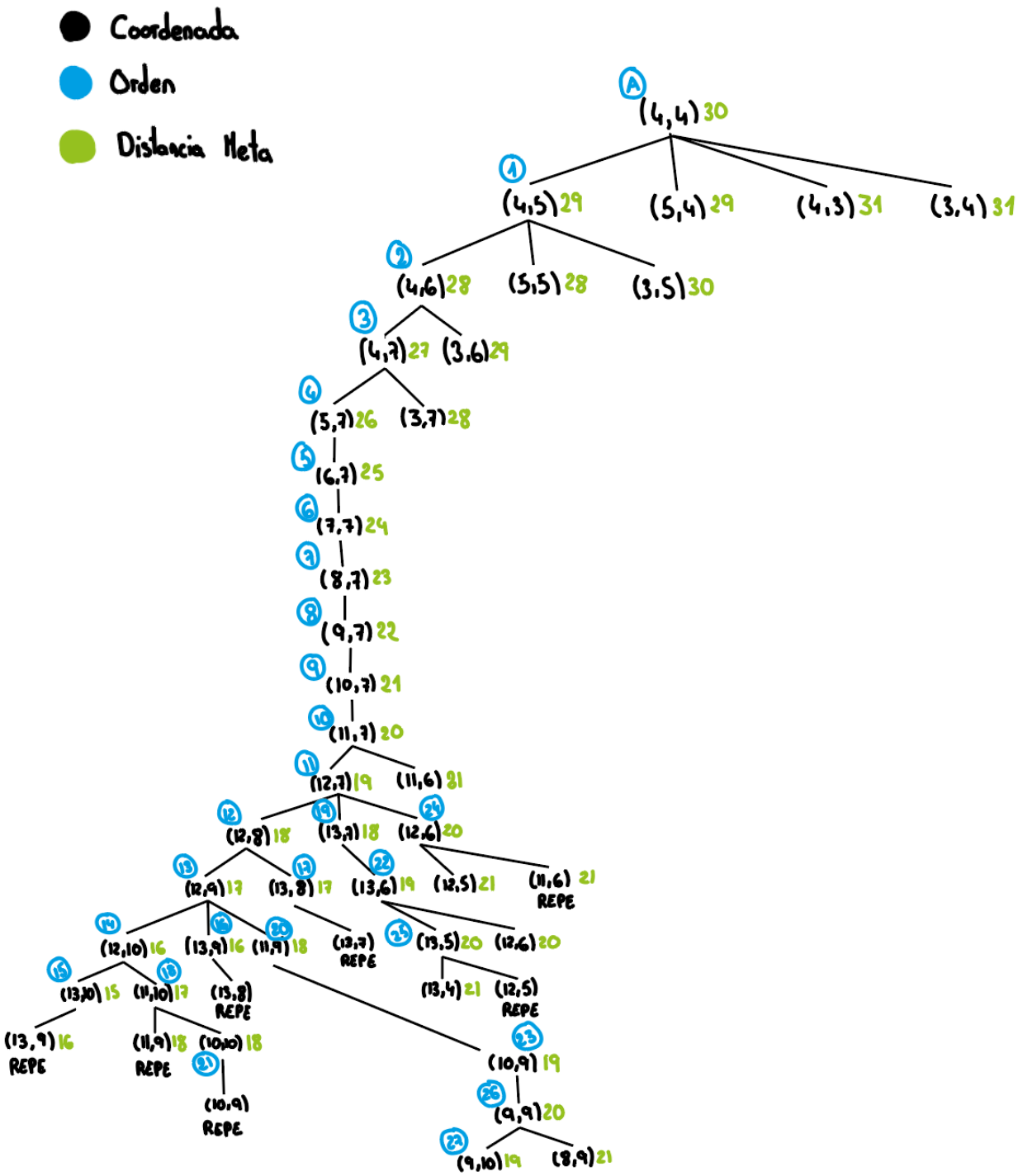


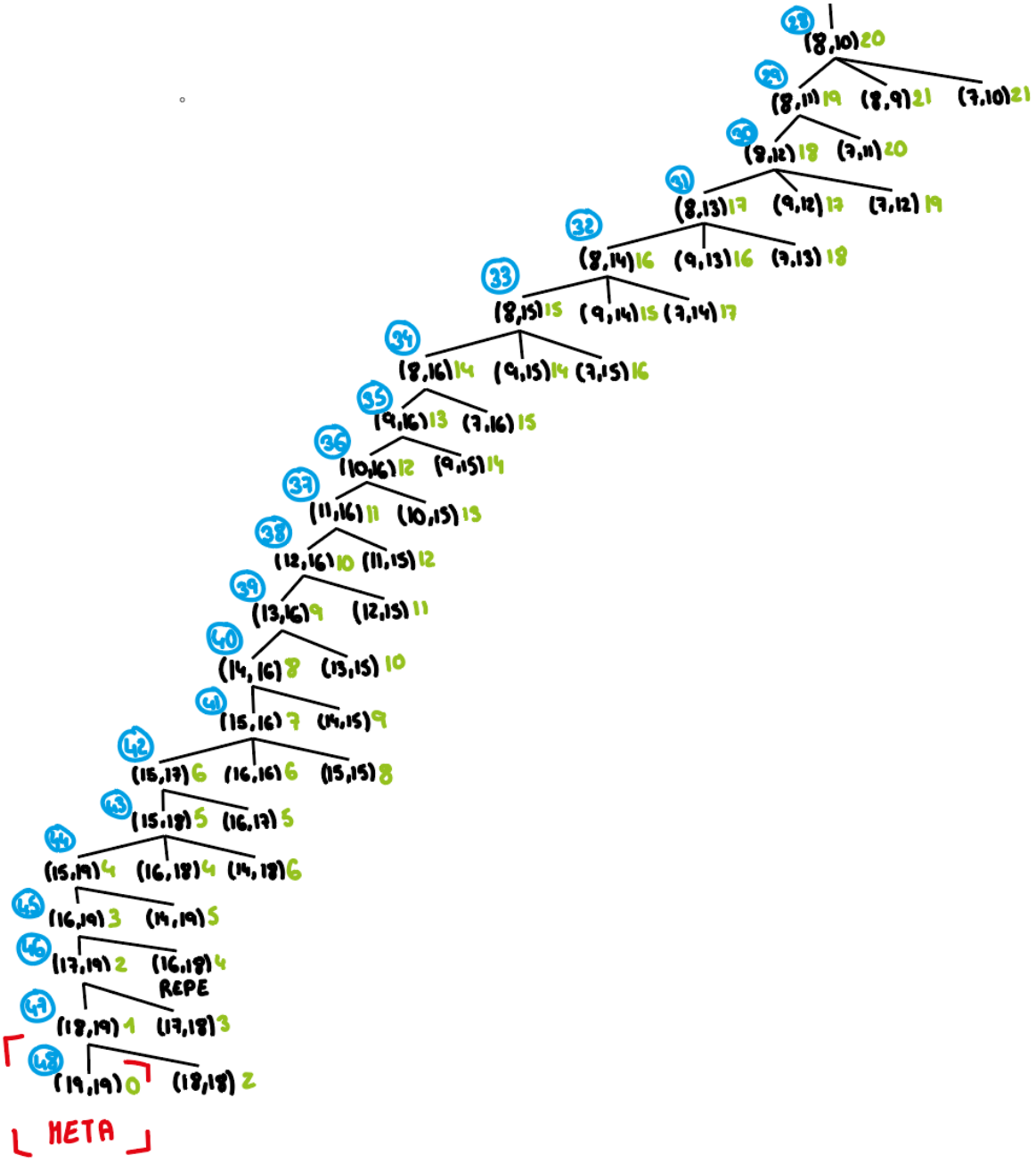
2.1.2 MAPA 2



2.2 ÁRBOL

Este árbol se corresponde con el del primer mapa. El recorrido se hizo solo con el valor de h* dado que al sumar la g de coste 1 e incrementándose por cada nivel el resultado al que se llega es el mismo.




(8,10)20
(8,11)19 (8,9)21 (7,10)21
(8,12)18 (7,11)20
(8,13)17 (9,12)17 (7,12)19
(8,14)16 (9,13)16 (7,13)18
(8,15)15 (9,14)15 (7,14)17
(8,16)14 (9,15)14 (7,15)16
(9,16)13 (7,16)15
(10,16)12 (9,15)14
(11,16)11 (10,15)13
(12,16)10 (11,15)12
(13,16)9 (12,15)11
(14,16)8 (13,15)10
(15,16)7 (14,15)9
(15,17)6 (16,16)6 (15,15)8
(15,18)5 (16,17)5
(15,19)4 (16,18)4 (14,18)6
(16,19)3 (14,19)5
(17,19)2 (16,18)4
REPE
(18,19)1 (17,18)3
(19,19)0 (18,18)2
META

2.3 LISTA

Lista del primer mapa.

1. [(4,4)30]

2. [(4,5)29, (5,4)29, (4,3)31, (3,4)31]

3. [(4,6)28, (5,5)28, (4,5)29, (5,4)29, (4,3)31, (3,4)31]

4. [(4,7)27, (5,5)28, (4,5)29, (5,4)29, (3,6)29, (4,3)31, (3,4)31]

5. [(5,7)26, (5,5) 28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

6. [(6,7)25, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

7. [(7,7)24, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

8. [(8,7)23, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

9. [(9,7)22, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

10. [(10,7)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

11. [(11,7)20, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

12. [(12,7)19, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

13. [(12,8)18, (13,7)18, (12,6)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

14. [(12,9)17, (13,8)17, (13,7)18, (12,6)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

15. [(12,10)16, (13,9)16, (13,8)17, (13,7)18, (11,9)18, (12,6)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

16. [(13,10)15, (13,9)16, (13,8)17, (11,10)17, (13,7)18, (11,9)18, (12,6)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

17. [((13,9)16, (13,8)17, (11,10)17, (13,7)18, (11,9)18, (12,6)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

18. [(13,8)17, (11,10)17, (13,7)18, (11,9)18, (12,6)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

19. [(11,10)17, (13,7)18, (11,9)18, (12,6)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

20. [(13,7)18, (11,9)18, (11,9)18, (10,10)18, (12,6)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

21. [(11,9)18, (10,10)18, (13,6)19, (12,6)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

22. [(10,10)18, (13,6)19, (10,9)19 (12,6)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

22. [(13,6)19, (10,9)19 (12,6)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

23. [(10,9)19 (12,6)20, (13,5)20, (12,6)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

24. [(12,6)20, (13,5)20, (12,6)20, (9,9)20, (11,6)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

25. [(13,5)20, (9,9)20, (11,6)21, (12,5)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

26. [(9,9)20, (11,6)21, (12,5)21, (13,4)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

27. [(9,10)19, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

28. [(8,10)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

29. [(8,11)19, (8,10)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

30. [(8,12)18, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

31. [(8,13)17, (9,12)17, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

32. [(8,14)16, (9,13)16, (9,12)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

33. [(8,15)15, (9,14)15, (9,13)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

34. [(8,16)14, (9,15)14, (9,14)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

35. [(9,16)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

36. [(10,16)12, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

37. [(11,16)11, (10,15)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

38. [(12,16)10, (11,15)12, (10,15)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

39. [(13,16)9, (12,15)11, (11,15)12, (10,15)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

40. [(14,16)8, (13,15)10, (12,15)11, (11,15)12, (10,15)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

41. [(15,16)7, (14,15)9, (13,15)10, (12,15)11, (11,15)12, (10,15)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

42. [(15,17)6, (16,16)6, (15,15)8, (14,15)9, (13,15)10, (12,15)11, (11,15)12, (10,15)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

43. [(15,18)5, (16,17)5, (16,16)6, (15,15)8, (14,15)9, (13,15)10, (12,15)11, (11,15)12, (10,15)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

44. [(15,19)4, (16,18)4, (16,17)5, (16,16)6, (14,18)6, (15,15)8, (14,15)9, (13,15)10, (12,15)11, (11,15)12, (10,15)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

45. [(16,19)3, (16,18)4, (16,17)5, (14,19)5, (16,16)6, (14,18)6, (15,15)8, (14,15)9, (13,15)10, (12,15)11, (11,15)12, (10,15)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

46. [(17,19)2, (16,18)4, (16,17)5, (14,19)5, (16,16)6, (14,18)6, (15,15)8, (14,15)9, (13,15)10, (12,15)11, (11,15)12, (10,15)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

47. [(18,19)1, (17,18)3, (16,18)4, (16,17)5, (14,19)5, (16,16)6, (14,18)6, (15,15)8, (14,15)9, (13,15)10, (12,15)11, (11,15)12, (10,15)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

48. [(19,19)0, (18,18)2, (17,18)3, (16,18)4, (16,17)5, (14,19)5, (16,16)6, (14,18)6, (15,15)8, (14,15)9, (13,15)10, (12,15)11, (11,15)12, (10,15)13, (9,15)14, (9,14)15, (7,16)15, (9,13)16, (7,15)16, (9,12)17, (7,14)17, (7,13)18, (7,12)19, (8,10)20, (7,11)20, (11,6)21, (12,5)21, (13,4)21, (8,9)21, (8,9)21, (7,10)21, (5,5)28, (3,7)28, (5,4)29, (3,6)29, (3,5)30, (4,3)31, (3,4)31]

3. DESARROLLO DE LA PRÁCTICA

3.1 BÚSQUEDA DEL CAMINO DE SALIDA

3.1.1 CÓDIGO CLASENODO

Creamos la clase nodo con su constructor y las funciones que nos dan la información de cada nodo, para poder evaluarlo y tenerlo en cuenta en el algoritmo o no.

```
Copyright

using System;
using System.Collections.Generic;
using Navigation.Interfaces;
using Navigation.World;



namespace Assets.Scripts.GrupoL
{
    33 referencias
    public class ClaseNodo
    {
        public CellInfo infoNodo; //Coordenadas de la celda en la que estamos
        public float h; //heuristica de cada nodo
        public ClaseNodo padre; //Nodo padre para poder asignarlo
        public float f;

        6 referencias
        public ClaseNodo(CellInfo celda) // Constructor de los nodos, pasamos de celda a nodo
        {
            this.infoNodo = new CellInfo(celda.x,celda.y);
            this.infoNodo.Type = celda.Type;

            this.padre = null; //Simplemente inicializamos los valores
            this.f=0;
            this.h = 0;
        }

        0 referencias
        public bool tienePadre() //Vemos si tiene padre o es Batman ;)
        {
            if(this.padre != null)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        public void ponerPadre(ClaseNodo PadreN) //Asignamos a cada nodo un padre
        {
            this.padre = PadreN;
        }
        2 referencias
        public ClaseNodo getPadre() {  // devuelve el Padre
            return this.padre;
        }

        2 referencias
        public bool esNodoIgual(ClaseNodo nodo) //Consultamos si 2 nodos son iguales
        {
            return this.infoNodo.Equals(nodo.infoNodo);
        }

        4 referencias
        public bool isWalkable() //Vemos si el nodo es andable (se ve con los tipos de celdas)
        {
            return this.infoNodo.Walkable;
        }

        1 referencia
        public float calcularHeuristica(ClaseNodo meta) //Calculamos nuestra heuristica, en nuestro caso es distancia Manhattan
        {
            return this.h = MathF.Abs(this.infoNodo.x - meta.infoNodo.x) + MathF.Abs(this.infoNodo.y - meta.infoNodo.y);
        }

        1 referencia
        public void calcularFNodo (ClaseNodo NodoMeta, float g) //Calcular f = g+h
        {
            this.f = g + calcularHeuristica(NodoMeta);
        }
    }
}
```

3.1.2 CÓDIGO AESTRELLA

Cálculo del camino por los nodos hasta llegar a la meta, utilizando el algoritmo A*.

```
using Assets.Scripts.GrupoL;
using Navigation.Interfaces;
using Navigation.World;
using System.Collections.Generic;
using UnityEngine;
using static Navigation.Interfaces.INavigationAlgorithm;


0 referencias
public class AEstrella : INavigationAlgorithm
{
    //Nodos necesarios
    ClaseNodo currentState;


    List<ClaseNodo> succesorStates = new List<ClaseNodo>();
    List<ClaseNodo> openList = new List<ClaseNodo>();
    List<ClaseNodo> closeList = new List<ClaseNodo>();

    //Variables necesarias
    private int g = 1; //coste que hemos puesto, constante
    private WorldInfo world;

    2 referencias
    public void Initialize(WorldInfo worldInfo, AllowedMovements allowedMovements)
    {
        world = worldInfo;
    }

    2 referencias
    public CellInfo[] GetPath(CellInfo NodoInicio, CellInfo NodoTarget) //Implementamos el algoritmo A*
    {
        CellInfo[] path = new CellInfo[1];
        //Convertimos las celdas en Nodos y obtenemos el nodo inicial y final
        ClaseNodo initialState = new ClaseNodo(NodoInicio);
        ClaseNodo goalState = new ClaseNodo(NodoTarget);

        openList.Add(initialState);
        while (openList.Count != 0)
        {
            currentState = openList[0];
            openList.RemoveAt(0);

            //Vemos si el nodo en el que estamos es el nodo meta
            if (currentState.esNodoIgual(goalState))
            {
                Debug.Log("Meta encontrada!!!!");
                path = CaminoRellenado(currentState).ToArray(); //Cogemos la rama en la que se llega a la meta
                openList.Clear();
                closeList.Clear();
                return path;

            }
            if (puedeExpandir(currentState))
            {
                expandirNodo(currentState);

                foreach (ClaseNodo sucessor in succesorStates)
                {
                    sucessor.ponerPadre(currentState);
                    sucessor.calcularFNodo(goalState, g);
                    openList.Add(sucessor);
                }
                ordenaHeuristica();
                closeList.Add(currentState);
                g++;
            }
            succesorStates.Clear();
        }
        return path;
    }
```

```
69      private List<CellInfo> CaminoRellenado (ClaseNodo nodoMeta)
70      {
71          List<CellInfo> aux = new List<CellInfo>();
72          ClaseNodo nodoPadre= nodoMeta.getPadre();
73
74          aux.Add(nodoMeta.infoNodo);
75
76          while(nodoPadre != null) //Cuando es null seria el nodo inicial
77          {
78              //Hacer una lista desde la meta hasta el origen
79              aux.Add (nodoPadre.infoNodo);
80              nodoPadre = nodoPadre.getPadre();
81          }
82          // Aqui se usa el reverse para hacer la lista en orden correcto desde origen a meta
83          aux.Reverse();
84          return aux;
85
86      }
87
        1 referencia
88      private bool puedeExpandir(ClaseNodo nodoToAdd) // Vemos si esta expandido el nodo
89      {
90          foreach (ClaseNodo nodo in closeList)
91          {
92              if (nodo.esNodoIgual(nodoToAdd))
93              {
94                  return false;
95              }
96
97          }
98
99              return true;
100
101     }
        1 rererencia
102     private void expandirNodo(ClaseNodo expNodo)
103     {
104         //Se definen las cuatro direcciones en las que se expandirán los nodos
105         ClaseNodo Up = new ClaseNodo(world[expNodo.infoNodo.x, expNodo.infoNodo.y - 1]);
106         ClaseNodo Down = new ClaseNodo(world[expNodo.infoNodo.x,expNodo.infoNodo.y + 1]);
107         ClaseNodo Right = new ClaseNodo(world[expNodo.infoNodo.x + 1, expNodo.infoNodo.y]);
108         ClaseNodo Left = new ClaseNodo(world[expNodo.infoNodo.x - 1, expNodo.infoNodo.y]);
109
110         // solo se añaden los nodos walkable
111         if(Up.isWalkable()) succesorStates.Add(Up);
112         if(Down.isWalkable())succesorStates.Add(Down);
113         if (Right.isWalkable()) succesorStates.Add(Right);
114         if (Left.isWalkable()) succesorStates.Add(Left);
115
116
117     }
118     //Se ordena de acuerdo al peso de cada nodo (ascendente debido al algoritmo)
        1 referencia
119     private void ordenaHeuristica()
120     {
121         for(int i=0;i<openList.Count;i++)
122         {
123             for(int j = 0; j < openList.Count-1; j++)
124             {
125                 if (openList[j].f > openList[j + 1].f)
126                 {
127                     ClaseNodo aux = openList[j];
128                     openList[j] = openList[j + 1];
129                     openList[j+1]= aux;
130                 }
131             }
132         }
133     }
134 }
135
```

3.2 BÚSQUEDA POR SUBOBJETIVOS

3.2.1 MAPA A SEGUIR

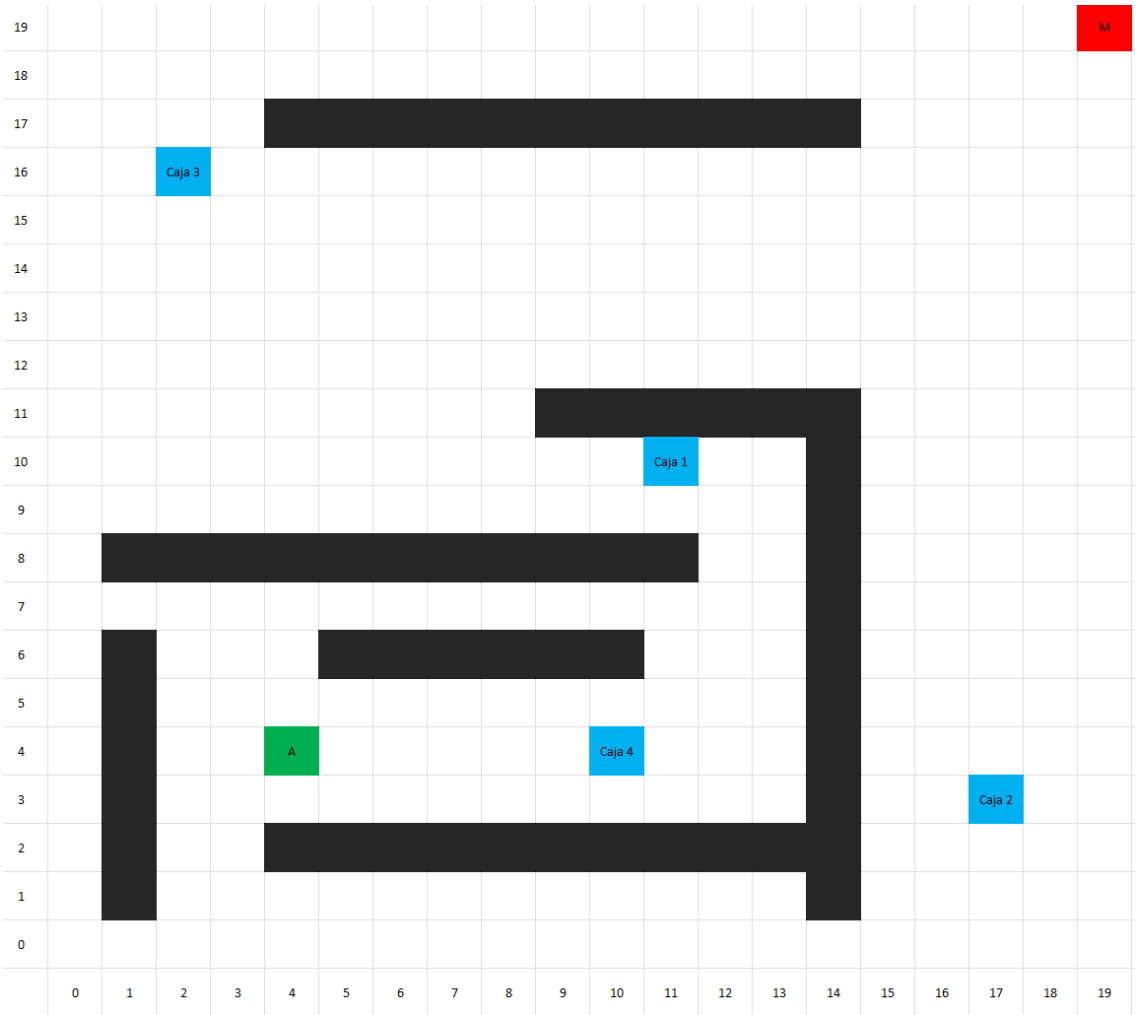
Aunque este recorrido **no sea el más eficiente** que se pueda conseguir, hemos establecido el orden de las cajas según su **distancia manhattan a la meta (priorizando el que esté más lejos primero)**. En el mapa que se muestra a continuación, se muestra el orden de recogida de cajas que se debe de seguir (caja4, caja3, caja2, caja1, M):

Caja 1 tiene una distancia manhattan de **17**

Caja 2 tiene una distancia manhattan de **18**

Caja 3 tiene una distancia manhattan de **20**

Caja 4 tiene una distancia manhattan de **24**



3.2.2 CÓDIGO BUSQUEDASUBOBJETIVO

```
using Assets.Scripts.GrupoL;
using Navigation.Interfaces;
using Navigation.World;
using System.Collections.Generic;
using UnityEngine;

1 referencia
public class BusquedaSubObjetivo : INavigationAgent
{
    4 referencias
    public CellInfo CurrentObjective { get; private set; }
    4 referencias
    public Vector3 CurrentDestination { get; private set; }
    3 referencias
    public int NumberOfDestinations { get; private set; }

    private WorldInfo _worldInfo;
    //Algoritmo que usamos, en nuestro caso A*
    private INavigationAlgorithm _navigationAlgorithm;

    //Nodo Origen (no solo es el nodo en donde empieza,
    //si no en donde esta cuando llega a un subobjetivo
    private ClaseNodo _origin;
    //lista de subobjetivos
    private List<ClaseNodo> _objectives;
    //Camino que recorre el agente
    private Queue<CellInfo> _path;


    0 referencias
    public BusquedaSubObjetivo(WorldInfo worldInfo, INavigationAlgorithm navigationAlgorithm, CellInfo PosicionInic) //constructor
    {
        // inicializamos el mundo
        _worldInfo = worldInfo;
        //asignamos el algoritmo a la variable
        _navigationAlgorithm = navigationAlgorithm;
        //asigno como origen la posición inicial
        _origin = new ClaseNodo(PosicionInic);

        _navigationAlgorithm.Initialize(_worldInfo, INavigationAlgorithm.AllowedMovements.FourDirections); //inicializo el algoritmo
    }

    2 referencias
    public Vector3? GetNextDestination()
    {
        //Si no hay ningun objetivo todavia, creamos la lista de subobjetivos
        if (_objectives == null)
        {
            _objectives = FillGoalsList();
        }
        //Si mi camino está vacio y mi lista de objetivos no esta vacia
        //voy buscando el camino a cada meta
        if ((_path == null || _path.Count == 0) && _objectives.Count > 0)
        {
            //ordeno la lista de metas poniendo el origen
            //(Recordemos que el origen depende de que subobjetivo tenemos siguiente)
            orderGoalList(_origin);
            //el objetivo actual es siempre la primera
            //posición de la lista de subobjetivos
            CurrentObjective = _objectives[0].infoNodo;
            //Y el número de destinos será el
            //tamaño de la lista de subobjetivos
            NumberOfDestinations = _objectives.Count;

            //Calculamos el camino usando el algoritmo base (en nuestro caso A*)
            CellInfo[] path = _navigationAlgorithm.GetPath(_origin.infoNodo, CurrentObjective);

            _path = new Queue<CellInfo>(path);

          //Cambiamos la meta una vez llegada a la anterior
            ChangeGoal();
        }

        if (_path.Count > 0)
        {

            CellInfo destination = _path.Dequeue();
            _origin.infoNodo = destination;
            CurrentDestination = _worldInfo.ToWorldPosition(destination);
        }
        return CurrentDestination;
    }
    //Simplemente eliminamos el primer subobjetivo de la lista
    //para que asi el siguiente subobjetivo sea el primero y no tener que hacer
    //codigo para que recorra la lista, simplemente leerá siempre la primera posición

```

```
1 referencia
private void ChangeGoal()
{
    _objectives.RemoveAt(0);
}

1 referencia
private List<ClaseNodo> FillGoalsList()
{
    //Creamos una lista auxiliar donde guardar los subobjetivos
    List<ClaseNodo> targets = new List<ClaseNodo>();
    for (int i = 0; i < _worldInfo.Targets.Length; i++)
    {
        //Guardamos los Subobjetivos (cofres)
        targets.Add(new ClaseNodo(_worldInfo.Targets[i]));
    }
    //Finalmente guardamos la meta
    targets.Add(new ClaseNodo(_worldInfo.Exit));
    //Devolvemos la lista con todos los Subobjetivos
    return targets;
}
//Se ordena la lista de subobjetivos menos la última posición (meta)
//de esta manera la meta final siempre será el Exit
1 referencia
private void orderGoalList(ClaseNodo  nodeActual)
{
    for (int j = 0; j < _objectives.Count - 1; j++)
    {
        for (int i = 0; i < _objectives.Count - 2; i++)
        {
            //Usamos para ordenar la lista de subobjetivos
            //la distancia manhattan de los cofres a la meta
            //y ponemos la lista en orden de manera que el primer
            //subobjetivo sea el cofre con la distancia manhattan más grande
            if (_objectives[i].calcularHeuristica(new ClaseNodo(_worldInfo.Exit)) < _objectives[i + 1].calcularHeuristica(new ClaseNodo(_worldInfo.Exit)))
            {
                ClaseNodo aux = _objectives[i];
                _objectives[i] = _objectives[i + 1];
                _objectives[i + 1] = aux;
            }
        }
    }
}

}
```