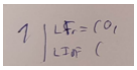
```
import pandas as pd
import matplotlib.pyplot as plt

mapa1 = pd.read_csv("https://docs.google.com/spreadsheets/d/1ZHxh7I_c-_DZ_3B99H_Ln-dCd85zO4WUNW5q2t0ndPk/export?format=csv&gid=0")

mapa5 = pd.read_csv("https://docs.google.com/spreadsheets/d/1afH_ZvYnWi18PaSiokSDW1wwmNM6piB0zwBiHpulnw8/export?format=csv&gid=0")

mapa6 = pd.read_csv("https://docs.google.com/spreadsheets/d/1QmVn5UoNXHwA77nC6O94MfBK576jBhr__Bnygnvv0Vc/export?format=csv&gid=0")

mapa7 = pd.read_csv("https://docs.google.com/spreadsheets/d/1e9L2fCICAZPASq2dm51y1KPUr1Z9caSOa1lrCQAGMn4/export?format=csv&gid=0")

mapa8 = pd.read_csv("https://docs.google.com/spreadsheets/d/1uLG8JvtE0QuygaNfkjNfady6HWs7qMc7moXB-tZuyGM/export?format=csv&gid=0")

mapa9 = pd.read_csv("https://docs.google.com/spreadsheets/d/15CeAb48U7LjTN57lP8eY5p_F8PpJzv-v-8wt5c0skyw/export?format=csv&gid=0")
```

ANÁLISIS DE LOS RESULTADOS

Análisis de la Traza:

Usaré este ejemplo dado que pese a su sencillez me sirve para explicar el algoritmo completo.

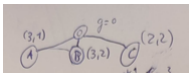
Para empezar, nuestro algoritmo usará dos listas, la primera ListaInterior comenzará vacía, y ListaFrontera que comenzará con el nodo de origen.



A partir de aquí, mientras listaFrontera no esté vacía, repetiremos la siguiente secuencia:

Primero, obtendremos el nodo de ListaFrontera con menor f(n), siempre cogeremos el primero, ya que a la hora de insertarlos lo haremos de forma ordenada. Después debemos comprobar si dicho nodo es meta, si no lo es lo moveremos de ListaFrontera a ListaInterior.

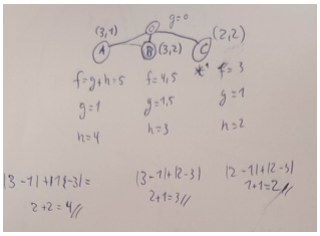
Una vez hecho esto, debemos comprobar los Nodos Hijos de el nodo seleccionado, en este caso los represento con A B y C.

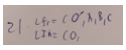


A partir de aquí llamaremos m al nodo hijo que estamos explorando para mayor comodidad.

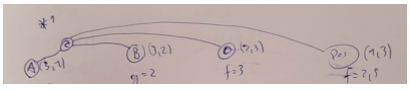
Luego, para cada hijo que no esté en listaInterior (para no repetir nodos ya explorados) debemos calcular g'(m). Esta es calculada sumando el valor de g del nodo padre más el coste del movimiento que se realiza para llegar al hijo, dicho coste se calcula como 1 en caso de movimientos verticales y horizontales y 1.5 para movimientos diagonales.

Tras esto, debemos comprobar si m se encuentra en listafrontera. Si no se encuentran en ella, podremos añadirlos calculando los valores de f, g, h y su nodo padre. El valor de h será calculado utilizando la heurística seleccionada (En el caso de esta traza utilizo Manhattan). Mientras que la f es calculada sumando los valores de g y h.

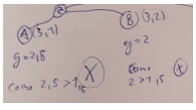




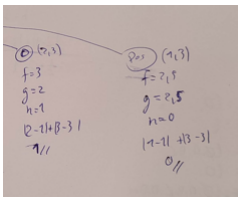
La siguiente iteración es sencilla, puesto que es repetir lo anterior, en este caso los hijos de C (el primer nodo en listafrontera ya que tiene menor f) son A,B,D y Des. Descartaremos Origen ya que se encuentra en ListaInterior. Por el resto, los cálculos y procedimientos son los mismos.

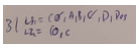


En esta iteración tanto A como B se encuentran en ListaFrontera, por lo que simplemente comprobaremos si los valores de g que acabamos de calcular son mejores a los almacenados, si esto es así los actualizaremos. En este caso no se cumple dicha premisa por lo que los dejaremos intactos e ignoraremos estos caminos, de esta forma nos aseguramos que el programa siempre busca un mejor camino, y nunca "empeora" uno ya almacenado.



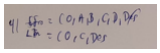
De la misma forma que antes D y Des no se encuentran en ListaFrontera, por lo que almacenaremos los valores comentados previamente y los añadiremos a la misma.





En esta iteración escogeremos el primer nodo de ListaFrontera (el que menor f(n) tiene). En este caso Des es un nodo meta, por lo que tan solo nos queda reconstruir un camino hacia la meta. Para ello utilizaremos los punteros hacia los padres de cada nodo cuando los guardabamos en ListaFrontera. Tras esto ya podemos salir del bucle con el camino (la g del último nodo incluyendo el coste del último movimiento) y el coste del mismo calculado.

Las listas quedan de la siguiente forma :



Por último, si en algún momento ListaFrontera quedase vacía, el algoritmo terminaría sin alcanzar una solución (camino) hacia el nodo meta.

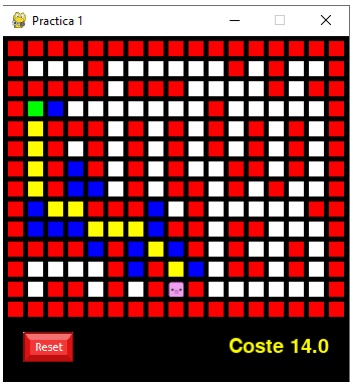
Distincciones entre Heurísticas

A continuación he creado algunos mapas de ejemplo para comprobar como se comporta el algoritmo utilizando distintas Heurísticas.

MAPA 1

Se ha guardado correctamente

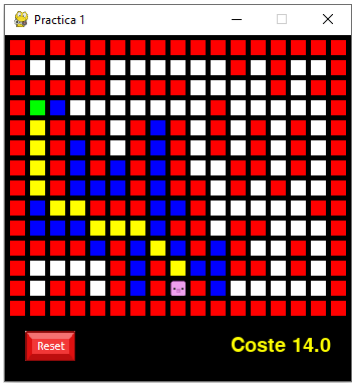
Nodos explorados: 28



EUCLÍDEA

Nodos del camino: 12

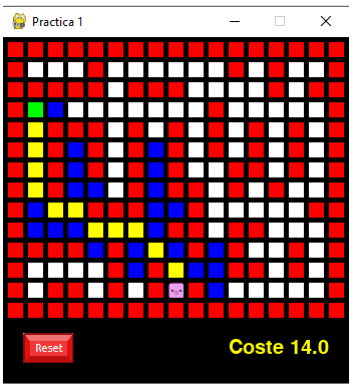
Nodos explorados: 40



DISTANCIA DIAGONAL

Nodos del camino: 12

Nodos explorados: 36

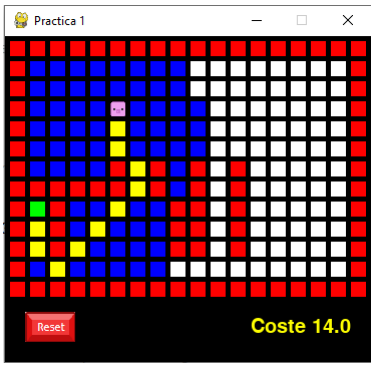


MAPA 5

MANHATTAN

Nodos del camino: 11

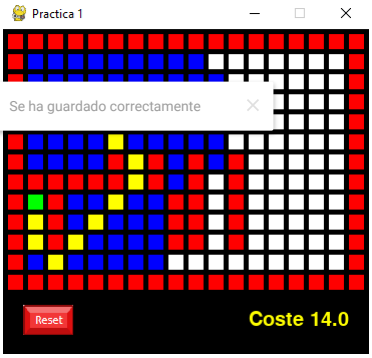
Nodos explorados: 76



EUCLIDEA

Nodos del camino: 11

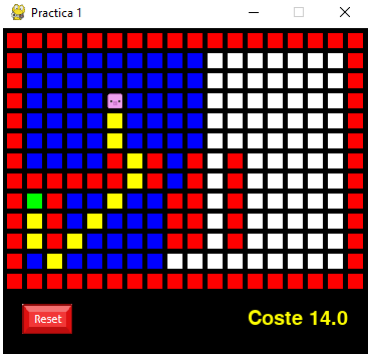
Nodos explorados: 83



DISTANCIA DIAGONAL

Nodos del camino: 11

Nodos explorados: 78

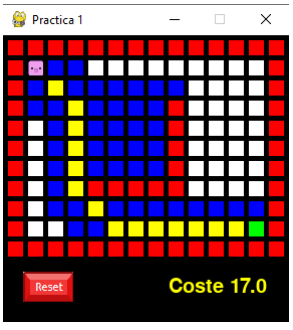


MAPA 6

MANHATTAN

Nodos del camino: 15

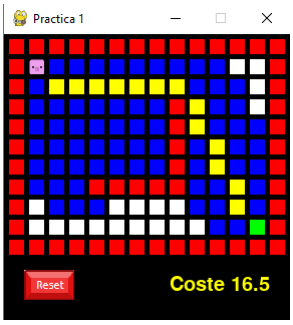
Nodos explorados: 59



EUCLIDEA

Nodos del camino: 14

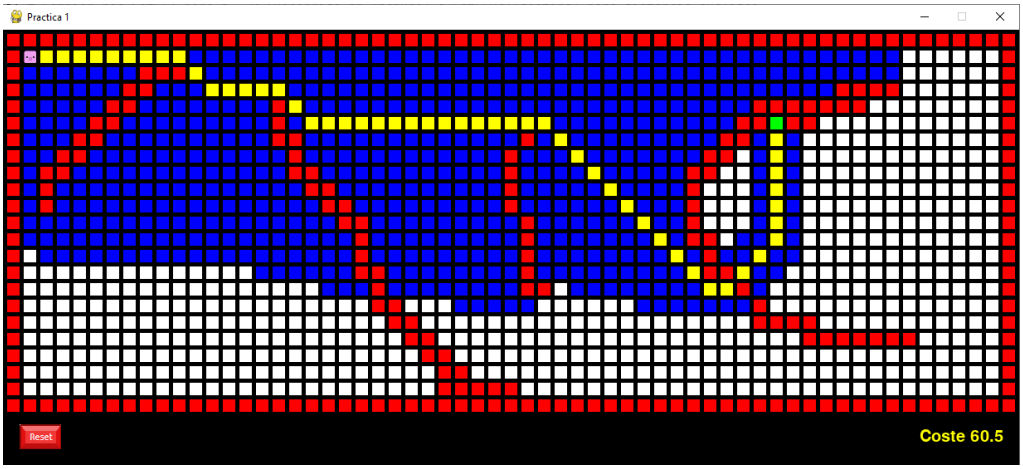
Nodos explorados: 81



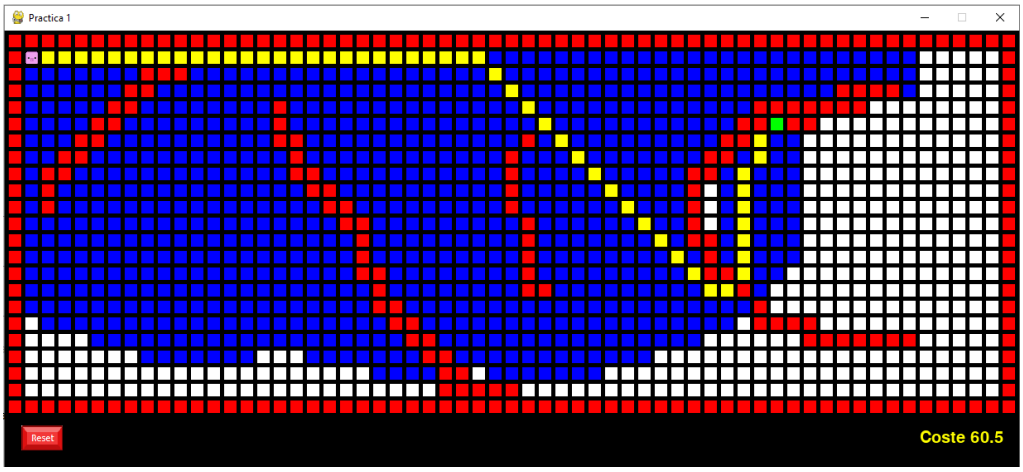
DISTANCIA DIAGONAL
Nodos del camino: 14
Nodos explorados: 75

MAPA 7

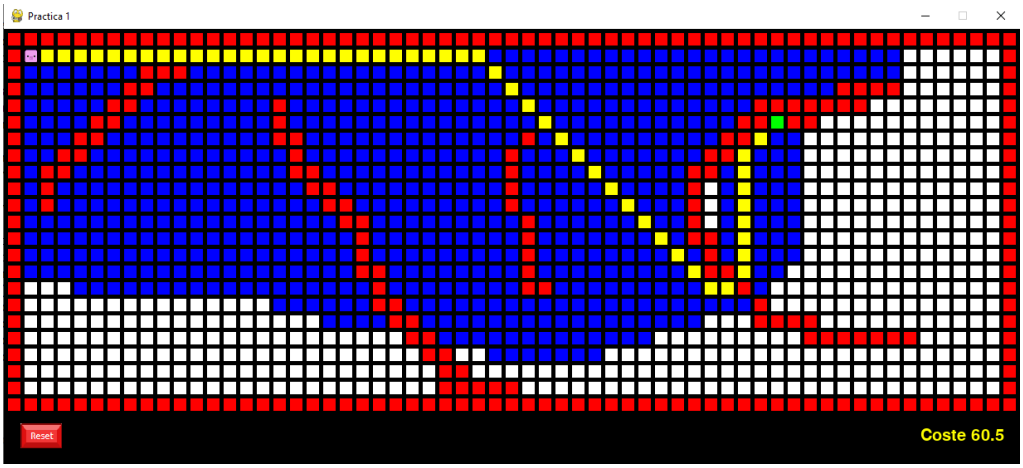
MANHATTAN
Nodos del camino: 52
Nodos explorados: 614



EUCLIDEA
Nodos del camino: 52
Nodos explorados: 803



DISTANCIA DIAGONAL
Nodos del camino: 52
Nodos explorados: 711

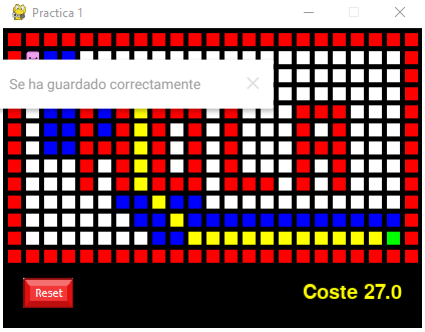


MAPA 8

MANHATTAN

Nodos del camino: 24

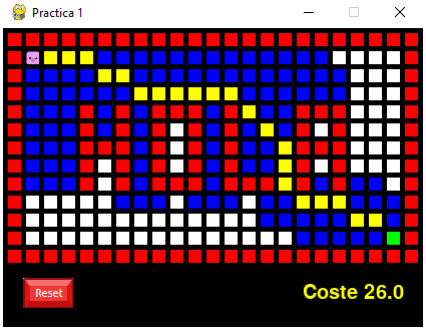
Nodos explorados: 65



EUCLIDEA

Nodos del camino: 22

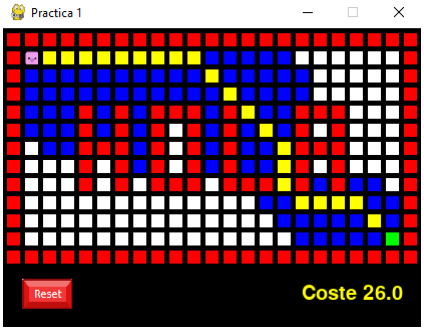
Nodos explorados: 124



DISTANCIA DIAGONAL

Nodos del camino: 22

Nodos explorados: 102

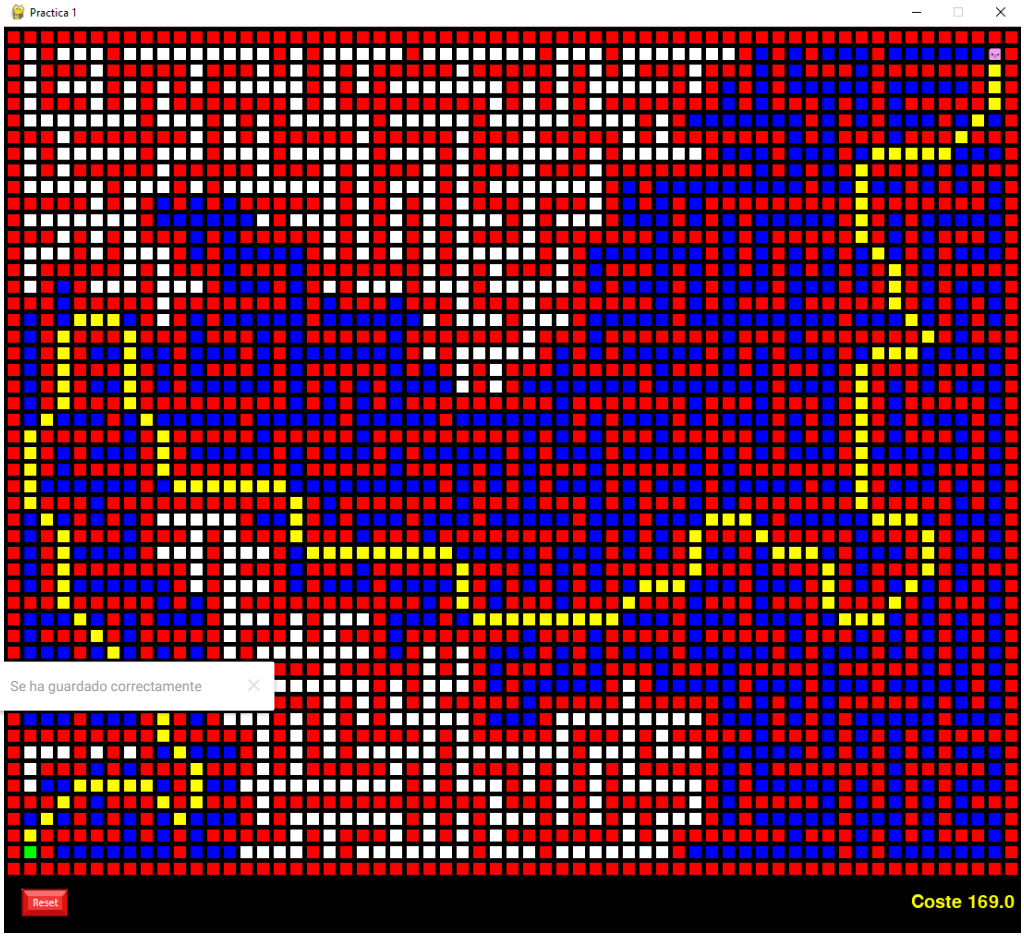


MAPA 9

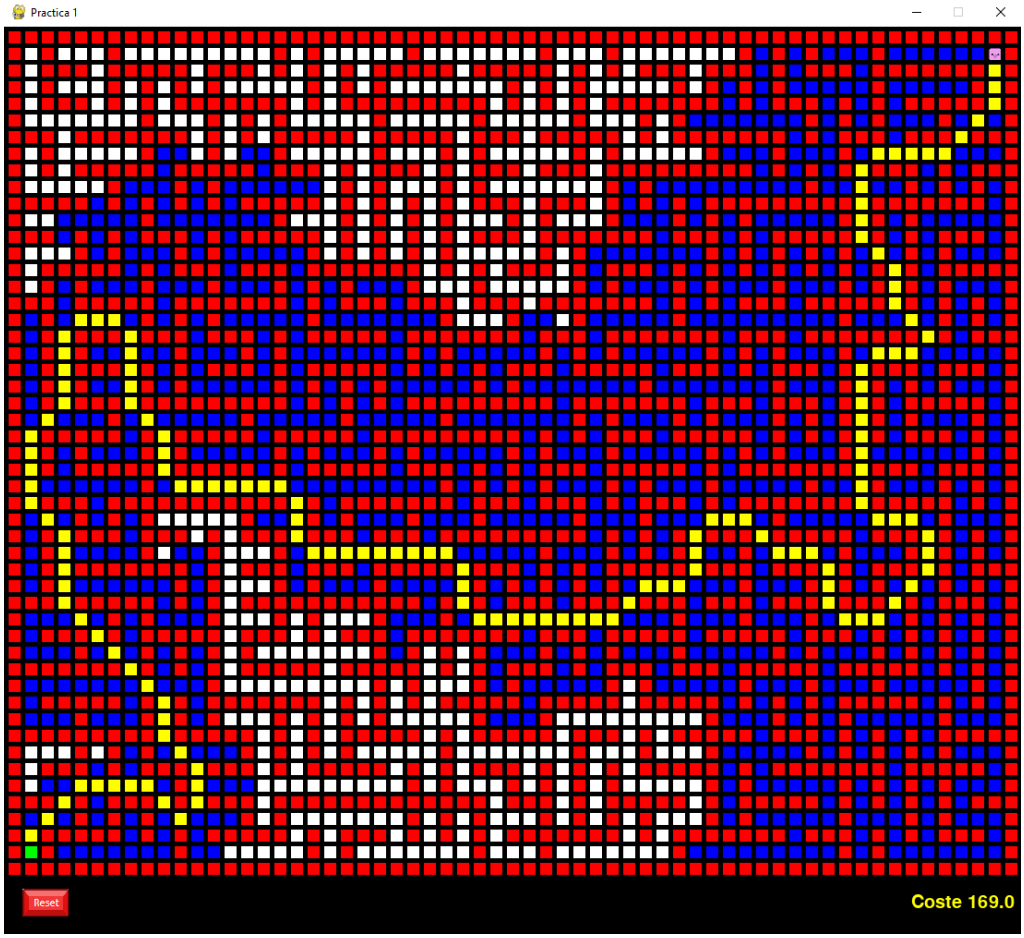
MANHATTAN

Nodos del camino: 144

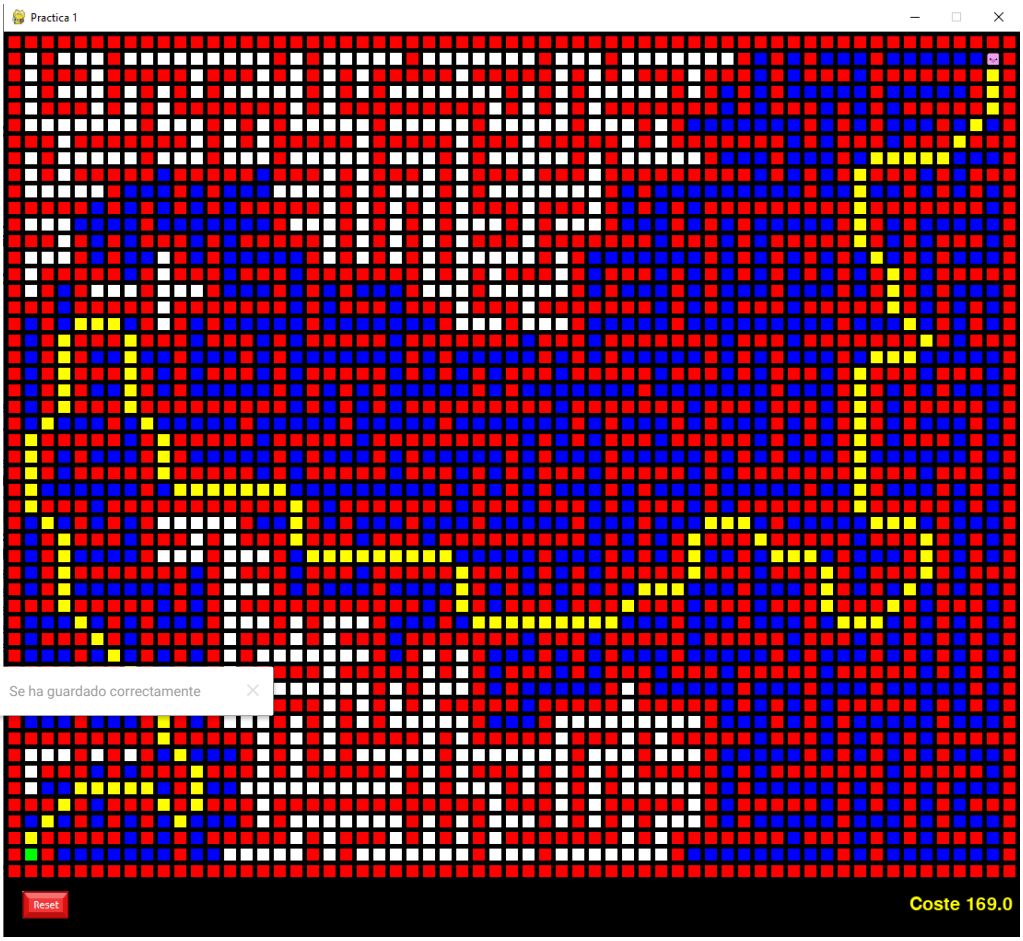
Nodos explorados: 994



EUCLIDEA
Nodos del camino: 144
Nodos explorados: 1058

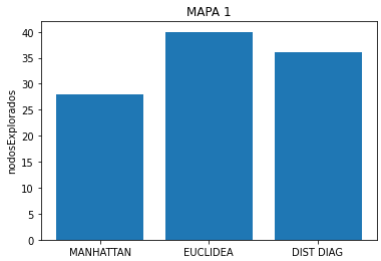


DISTANCIA DIAGONAL
Nodos del camino: 144
Nodos explorados: 1033



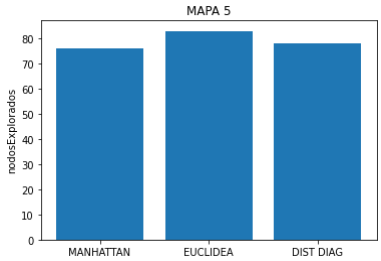

```
plt.title("MAPA 1")
plt.ylabel("nodosExplorados")
plt.bar(mapa1.HEURISTICA, mapa1.EXPLORADOS)
```

```
<BarContainer object of 3 artists>
```



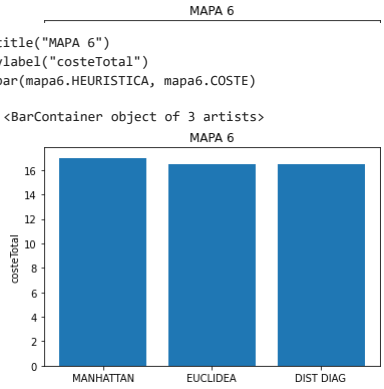

```
plt.title("MAPA 5")
plt.ylabel("nodosExplorados")
plt.bar(mapa5.HEURISTICA, mapa5.EXPLORADOS)
```

```
<BarContainer object of 3 artists>
```




```
plt.title("MAPA 6")
plt.ylabel("nodosExplorados")
plt.bar(mapa6.HEURISTICA, mapa6.EXPLORADOS)
```

```
<BarContainer object of 3 artists>
```



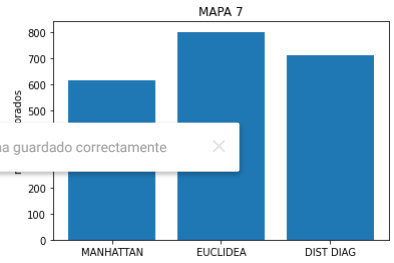

```
plt.title("MAPA 6")
plt.ylabel("costeTotal")
plt.bar(mapa6.HEURISTICA, mapa6.COSTE)
```

```
<BarContainer object of 3 artists>
```



```
plt.title("MAPA 7")
plt.ylabel("nodosExplorados")
plt.bar(mapa7.HEURISTICA, mapa7.EXPLORADOS)
```

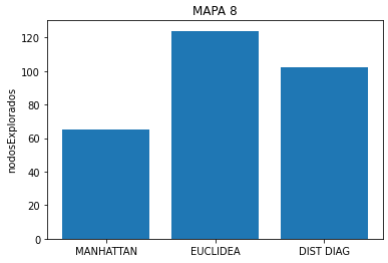
```
<BarContainer object of 3 artists>
```



Se ha guardado correctamente

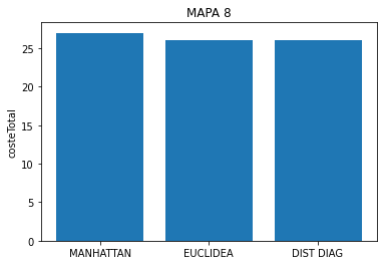
```
plt.title("MAPA 8")
plt.ylabel("nodosExplorados")
plt.bar(mapa8.HEURISTICA, mapa8.EXPLORADOS)
```

```
<BarContainer object of 3 artists>
```



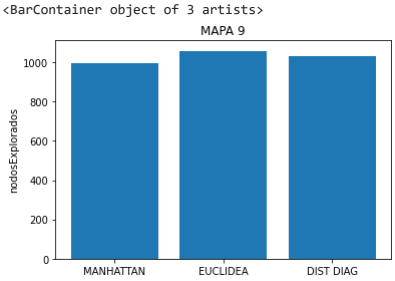

```
plt.title("MAPA 8")
plt.ylabel("costeTotal")
plt.bar(mapa8.HEURISTICA, mapa8.COSTE)
```

```
<BarContainer object of 3 artists>
```




```
plt.title("MAPA 9")
plt.ylabel("nodosExplorados")
plt.bar(mapa9.HEURISTICA, mapa9.EXPLORADOS)
```

```
<BarContainer object of 3 artists>
```



Una vez visto el resultado utilizando distintas heurísticas en los mismos mapas y analizado los resultado mediante gráficos de barras podemos sacar varias conclusiones. Para empezar y muy importante, podemos afirmar que la heurística Manhattan NO es admisible, esto se puede

observar en los resultados obtenidos en coste del *mapa 6* donde el coste empleando esta heurística es 0.5 mayor o en el *mapa 8* donde el coste llega a ser 1.0 mayor con respecto a las otras dos heurísticas.

Una vez hecho este análisis y si tuviese que escoger una heurística, utilizaría la Distancia Diagonal, puesto que ofrece mejores resultados entre las dos heurísticas admisibles. Y entre las dos restantes Manhattan supone un menor número de nodos explorados lo que conlleva un menor

Funciones Auxiliares e Información Relevante

Funciones Auxiliares

Por último me gustaría añadir algunas funciones propias que he añadido para facilitarme el desarrollo del algoritmo.

Por un lado tengo ConstruyeCamino. Al cual le paso un nodo (meta) y el camino y recorre de forma recursiva los nodos de padre en padre hasta llegar al inicio para reconstruir el camino.

```
def construyecamino(nodo,camino):
    if nodo.GetPadre() == None:   # Es el inicio
        camino[nodo.GetX()][nodo.GetY()] = 'y'
    else:
        camino[nodo.GetX()][nodo.GetY()] = 'y'
        construyecamino(nodo.GetPadre(),camino)
```

Luego Utilizo algunas funciones auxiliares dentro de la clase Nodo, para facilitarme los cálculos:

```
 @staticmethod
    def Manhattan(c1,c2): # Heurística Manhattan
        return abs(c2.getFila()-c1.getFila())+abs(c2.getCol()-c1.getCol())
    @staticmethod
    def Euclidea(c1,c2): # Heurística Euclídea
        return math.sqrt(pow(c2.getFila()-c1.getFila(),2)+pow(c2.getCol()-c1.getCol(),2))
```

Se ha guardado correctamente

```
                                Distancia Diagonal
        dy = abs(c1.getCol() - c2.getCol())
        return ((dx + dy) + (1.5 - 2 * 1) * min(dx, dy))
     @staticmethod
    def CalculaCoste(m,n):
        coste = 1.0
        if m.getFila() != n.getFila() and m.getCol() != n.getCol(): # Movimiento diagonal
            coste = 1.5
```

Dibujar Vecinos de Azul

He modificado algunas partes del main, de forma que puedo mostrar los nodos que se exploran de color azul y así dejar el análisis más visual. Para ello simplemente he añadido una opción más en el if que comprueba los caracteres de camino.

```
elif camino[fil][col]=='/':
                        pygame.draw.rect(screen, AZUL, [(TAM+MARGEN)*col+MARGEN, (TAM+MARGEN)*fil+MARGEN, TAM, TAM], 0)
```

De esta forma para marcarlos simplemente tengo que añadir este carácter a las casillas que añado en el cálculo de los Nodos Vecinos.

```
def vecinos(mapi,node,camino):
    veins = []
    alcanzados = 0
    for x in range(-1,2):     # Recorro las 9 Direcciones (incluyendo la misma casilla)
        for y in range(-1,2):
            if not (x == 0 and y == 0):
                cx = node.GetX()+x
                cy = node.GetY()+y
                if cx > 0 and cx < mapi.getAlto()-1 and cy > 0 and cy < mapi.getAncho()-1:
                    if mapi.getCelda(cx,cy) == 0:
                        veins.append(Casilla(cx,cy))
                        #camino[cx][cy] = '/'  # <---- Descomentar para Pintar Los vecinos del Nodo de Azul
```

Mapas Grandes en Pantalla

Algunos mapas de los que he creado son demasiado grandes para mostrarlos con el tamaño de ventana por defecto, por lo que se mostrarán parcialmente. Para solucionar esto simplemente he cambiado la varibale global *TAM* y de esta manera ajustar la ventana para que se muestre todo el mapa.

Cambiar entre Heurísticas

Para Cambiar de heurística he marcado en el código mediante comentarios las zonas de código a cambiar. Sin embargo este es un ejemplo de cómo usar la heurística Distancia Diagonal y Manhattan. Dentro de la función *aEstrella*:

```
 for hijo in hijos:
                m = Nodo(hijo.getFila(),hijo.getCol(),n,Nodo.CalculaCoste(hijo,Casilla(n.GetX(),n.GetY()))) #Creo un nodo m con la casilla hijo
                m.g = n.g + m.coste
                m.SetFDDiag(destino) # Cambiar por SetF() para h = 0 (Recorrido en Anchura), SetFMan(destino) para Manhattan, SetFEucl(destino) para distancia euclídea o SetFDDiag(des

                existeint = False
                existefront = False
                for l in listaint:   # Compruebo que no existe en lista interior
                    if l.GetX() == m.GetX() and l.GetY() == m.GetY():
                        existeint = True
                for l in listafront: # Compruebo que no existe en lista frontera, si existe me guardo el índice para compararlo y modificarlo
                    if l.GetX() == m.GetX() and l.GetY() == m.GetY():
                        existefront = True
                        index = listafront.index(l)
                if not existeint:  # Si el hijo NO está en lista interior:
                    if not existefront:     # m no está en lista frontera
                        bisect.insort(listafront,m)
                        #InsOrden(listafront,m)       #Inserto en orden el elemento f >
                    elif m.Getg() < listafront[index].Getg():   # Si la nueva g de m es menor a la almacenada (nuevo camino + barato) cambio el padre y recalculo F
                        listafront[index].SetPadre(n)
                        listafront[index].SetCoste(Nodo.CalculaCoste(Casilla(n.GetX(),n.GetY()),Casilla(m.GetX(),m.GetY()))) #Recalculo Coste
```

```
                    listafront[index].g = n.g + listafront[index].coste
                    listafront[index].SetFDDiag(destino) # Cambiar por SetF() para h = 0 (Recorrido en Anchura), SetFMan(destino) para Manhattan, SetFEucl(destino) para distancia
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-966622406c29> in <module>()
----> 1 for hijo in hijos:
      2                 m =
Nodo(hijo.getFila(),hijo.getCol(),n,Nodo.CalculaCoste(hijo,Casilla(n.GetX(),n.GetY()))) #Creo un nodo
m con la casilla hijo
      3                 m.g = n.g + m.coste
      4                 m.SetFDDiag(destino) # Cambiar por SetF() para h = 0 (Recorrido en Anchura),
SetFMan(destino) para Manhattan, SetFEucl(destino) para distancia euclídea o SetFDDiag(destino) para
distancia diagonal
      5

NameError: name 'hijos' is not defined
```

```
for hijo in hijos:
                m = Nodo(hijo.getFila(),hijo.getCol(),n,Nodo.CalculaCoste(hijo,Casilla(n.GetX(),n.GetY()))) #Creo un nodo m con la casilla hijo
                m.g = n.g + m.coste
                m.SetFMan(destino) # Cambiar por SetF() para h = 0 (Recorrido en Anchura), SetFMan(destino) para Manhattan, SetFEucl(destino) para distancia euclídea o SetFDDiag(desti

                existeint = False
                existefront = False
                for l in listaint:   # Compruebo que no existe en lista interior
                    if l.GetX() == m.GetX() and l.GetY() == m.GetY():
                        existeint = True
                for l in listafront: # Compruebo que no existe en lista frontera, si existe me guardo el índice para compararlo y modificarlo
                    if l.GetX() == m.GetX() and l.GetY() == m.GetY():
                        existefront = True
                        index = listafront.index(l)
                if not existeint:  # Si el hijo NO está en lista interior:
                    if not existefront:     # m no está en lista frontera
                        t(listafront,m)
                        istafront,m)       #Inserto en orden el elemento f >
                    elif m.getg() < listafront[index].Getg():   # Si la nueva g de m es menor a la almacenada (nuevo camino + barato) cambio el padre y recalculo F
                        listafront[index].SetPadre(n)
                        listafront[index].SetCoste(Nodo.CalculaCoste(Casilla(n.GetX(),n.GetY()),Casilla(m.GetX(),m.GetY()))) #Recalculo Coste
                        listafront[index].g = n.g + listafront[index].coste
                        listafront[index].SetFMan(destino) # Cambiar por SetF() para h = 0 (Recorrido en Anchura), SetFMan(destino) para Manhattan, SetFEucl(destino) para distancia eu
```

Se ha guardado correctamente

Tal y como el comentario explica las distintas heurísticas se utilizan cambiando esas 2 líneas por:

1. **Recorrido en Anchura (h = 0)** -> *SetF()*
2. **Manhattan** -> *SetFMan(destino)*
3. **Euclídea** -> *SetFEucl(destino)*
4. **Distancia Diagonal** -> *SetFDDiag(destino)*

0 s completado a las 18:12