Primer proyecto de búsquedas no informadas

Juan Camilo Azcarate Cardenas - 201958932
Estefany Castro Agudelo - 201958552
Daniel Gomez Suarez - 201958732
Valentina Hurtado Garcia - 201958542

Universidad del valle - Tulua(Valle)
Facultad de ingeniería
Introducción a la inteligencia artificial
Joshua David Triana Madrid
2023

1. Introducción

El propósito de este documento es exponer la solución de algoritmos búsqueda no informada representados mediante árboles vistos en clase, los cuales son: Preferente por amplitud, de costo uniforme y preferente por profundidad.

El documento contará con una breve explicación, funcionamiento e imágenes de los algoritmos empleados, cuya implementación fue en el lenguaje de programación Python.

2. Librerías

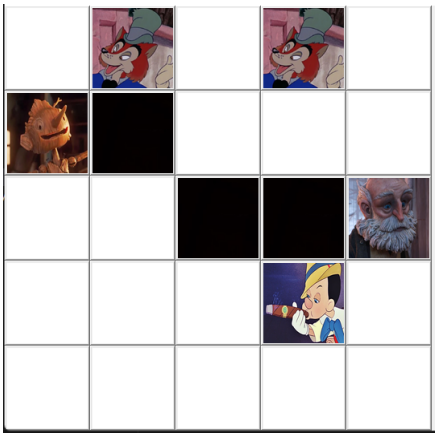
2.1. ***Librería numpy:*** Librería de Python especializada en el cálculo y el análisis de datos, incluyendo objetos llamados arrays que permiten representar colecciones de datos de un mismo tipo en varias dimensiones y funciones eficientes para su manipulación, siendo esta la razón de selección de uso de esta librería.

2.2. ***Librería custom tkinter:*** Librería de Python destinada a la creación y el desarrollo de aplicaciones de escritorio, facilitando el desarrollo de una interfaz gráfica, siendo esta usada como puente entre la información suministrada por el usuario y el software.

2.3. ***Librería pillow:*** Es una biblioteca adicional gratuita y de código abierto para el lenguaje de programación Python que agrega soporte para abrir, manipular y guardar muchos formatos de archivo de imagen diferentes.

3. Implementación

3.1. ***Definición de matriz por txt:*** 0 = casilla en blanco, 1 = pinocho, 2 = Gepetto, 3 = cigarrillo, 4 = zorro, 5 = bloque.



File: Mapas/mapa-default.txt

```
1  5 5
2  0 4 0 4 0
3  0 5 0 0 0
4  1 0 5 5 2
5  0 0 0 3 0
6  0 0 0 0 0
```

3.2. Clases

3.2.1. *Clase nodo*: En la cual se maneja el contenido de cada nodo.

3.2.2. *Clase posición:* La cual es usado nodo.pos para representar las coordenadas x,y de los nodos.

```
class nodo: #Creamos la clase nodo.
    def __init__(self, pos, recorridos, camino, costo):
      self.pos = pos
      self.recorridos = recorridos
      self.camino = camino
      self.costo = costo

class posicion:#creamos la clase posicion
    def __init__(self, posx, posy):
      self.posx = posx
      self.posy = posy
```

3.3. Funciones

3.3.1. *Definición costos:* La usamos para extraer el costo en nodo de la matriz.

```
def costof(matriz,x,y):
  valor = matriz[x][y]
  if valor == 0 or valor == 1 or valor == 2:
   return 1
  elif valor == 3:
   return 2
  elif valor == 4:
   return 3
  elif valor == 5:
    return 4
  elif valor == None:
   return 5
```

3.3.2. *Definición buscar elemento:* Creamos la función buscar elemento para ver si se encuentra en la lista el elemento.

```
def buscarElemento(lista, elemento):
    for i in lista:
      if i.posx == elemento.posx and i.posy == elemento.posy:
        return True

    else:
      return False
```

3.4. Definición búsqueda profundidad: La cual mediante la matriz, piso y posición inicial.

```
def busquedaProfundida(matriz,x,y,piso,nodos):
    juego = np.loadtxt( matriz , skiprows = 0) #Declaramos a como el archivo.
    pila = []
    inicio = nodo(posicion(x,y),[],[posicion(x,y)],0)
    nodoc = nodos
    pila.append(inicio)#Utilizamos la posicion inicial del robot.
```

Mediante un bucle y la extracción del nodo más profundo de la pila evaluamos

```
#Busqueda amplitud
while(True):

 #paro si no encontré a gepetto
  if len(pila) == 0:
    lista = []
    costot = nodoActual.costo
    for i in nodoActual.camino:
     lista.append((i.posx,i.posy))
    return (1,lista,costot,nodoc)

  nodoActual = pila.pop(-1)#sacamos el nodo mas profundo de la cola y sus respectivas posiciones x,y
  posX = nodoActual.pos.posx
  posY = nodoActual.pos.posy
```

Si encontramos a gepetto nos detenemos y enviamos resultados

```
#paro si encontré a gepetto
if juego[posX][posY] == 2:
  lista = []
  costot = nodoActual.costo
  for i in nodoActual.camino:
   lista.append((i.posx,i.posy))
  return (2,lista,costot,nodoc)
```

Evitamos los muros catalogados como dato numero 5

```
if not(juego[posX][posY] == 5):
```

Evaluamos los datos para cada dirección y los posibles datos para las mismas, además de verificar que la misma sea válida en la matriz-

```
 #Arriba
 if(nodoActual.pos.posy>0):#verificamos que no se salga de los limites de la matriz
   posicionNueva = posicion(nodoActual.pos.posx,nodoActual.pos.posy-1)
   recorridoA = nodoActual.recorridos.copy()
   caminoA = nodoActual.camino.copy()
   nodoc = nodoc + 1
```

En caso de que sea válida verificamos que no haya sido un nodo recorrido,si cumple enviamos el nuevo nodo con los datos.

```
#primero declaramos una copia de las funciones si cumple que no se a hecho antes modifca los datos
#y se agregan los caminos,ademas de sumar a la cola
  if buscarElemento(nodoActual.recorridos,posicionNueva)==False and juego[posicionNueva.posx
  ][posicionNueva.posy]!=1:
    recorridoA.append(nodoActual.pos)
    caminoA.append(posicionNueva)
    costoa = nodoActual.costo + costof(juego,nodoActual.pos.posx,nodoActual.pos.posy-1)
    nuevoNodo = nodo(posicionNueva,recorridoA,caminoA,costoa)
    pila.append(nuevoNodo)
```

3.5. Modificaciones profundidad iterativa: En el caso de profundidad iterativa agregamos al bucle la confirmación de que no nos pasemos en el nivel del nodo.

```
lista = []
for i in nodoActual.camino: #verificamos que no se pase del piso limite
  lista.append((i.posx,i.posy)) #en caso contrario para
if len(lista) > piso:
  return (False,lista,nodoc)
```

En caso contrario le enviamos los datos a otra función propia de este algoritmo.

3.6. Definición profundidad: La cual al verificar que llegó al piso lo re-envía con uno mayor hasta encontrar el objetivo.

```
def profundidadi(matriz,x,y):
 piso = 0
 nodos = 0
 while (True):
  resultado= []
  resultado= busquedaProfundida(matriz,x,y,piso,nodos)
  nodos = resultado[2]
                                  #declaramos resultado para que le de un tope a busqueda profundida
  if resultado[0] == False:  #y si no encuentra la solucion siga bsucando con un piso mayor
   piso = piso + 1

  elif resultado[0] == 1: #encontre a gepetto
   return False

  elif resultado[0] == 2: #no encontre a gepetto
   return resultado[1]
```

3.7. Modificaciones para amplitud: En amplitud en vez de trabajar con el más profundo de la pila trabajamos con el primero de la cola, y hacemos rotación de la misma si se cambia de nivel.

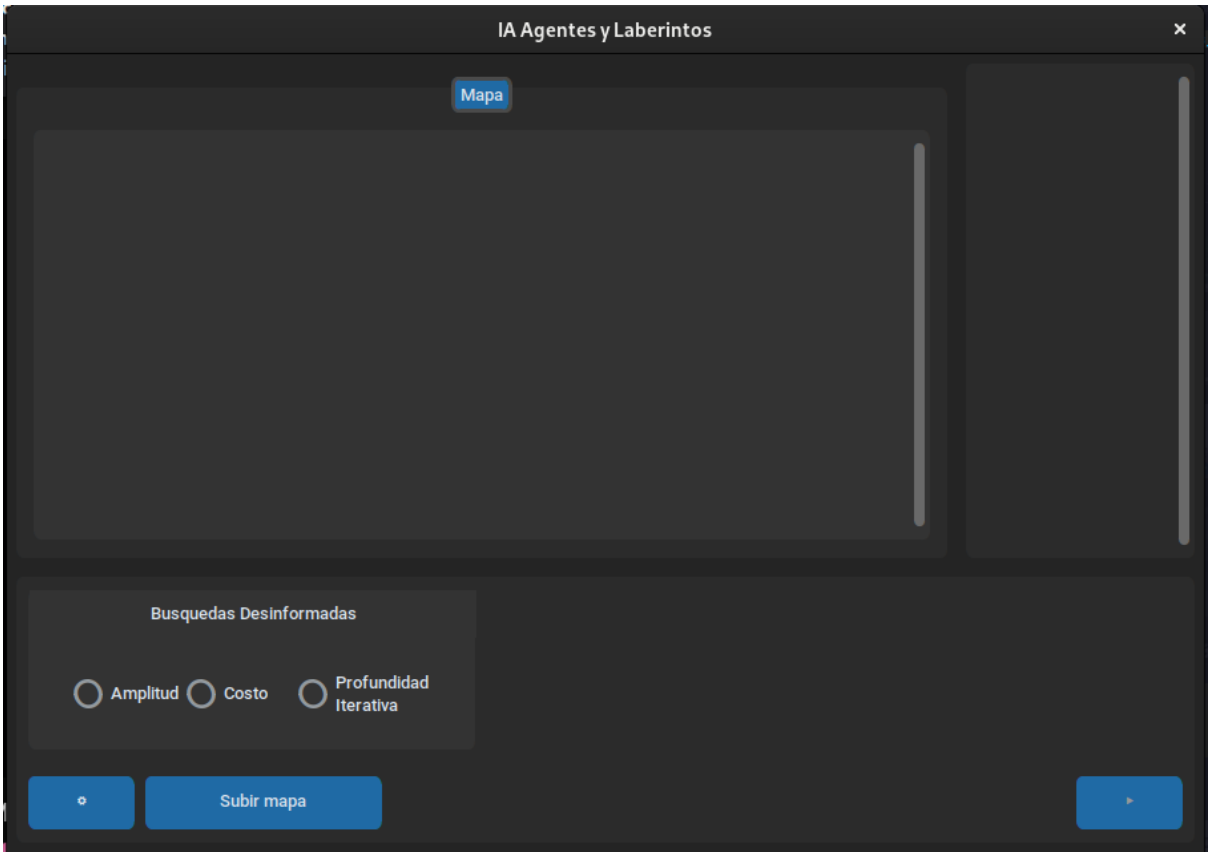
```
nodoActual = cola[0] #en este punto revisamos el nodo actual
contador = ((len(nodoActual.camino)% 2)) +1 #sacamos modulo saldra entre 1 y 2
if contador == 1 and nodoanterior == 2:#si es 1 y cambio volteamos la cola
 cola = cola[::-1]                    #si fuera iquierda a derecha se voltea y viceversa
 nodoanterior = 1
elif contador == 2 and nodoanterior == 1:#si es 2 y cambio volteamos la cola
 cola = cola[::-1]
 nodoanterior = 2
```

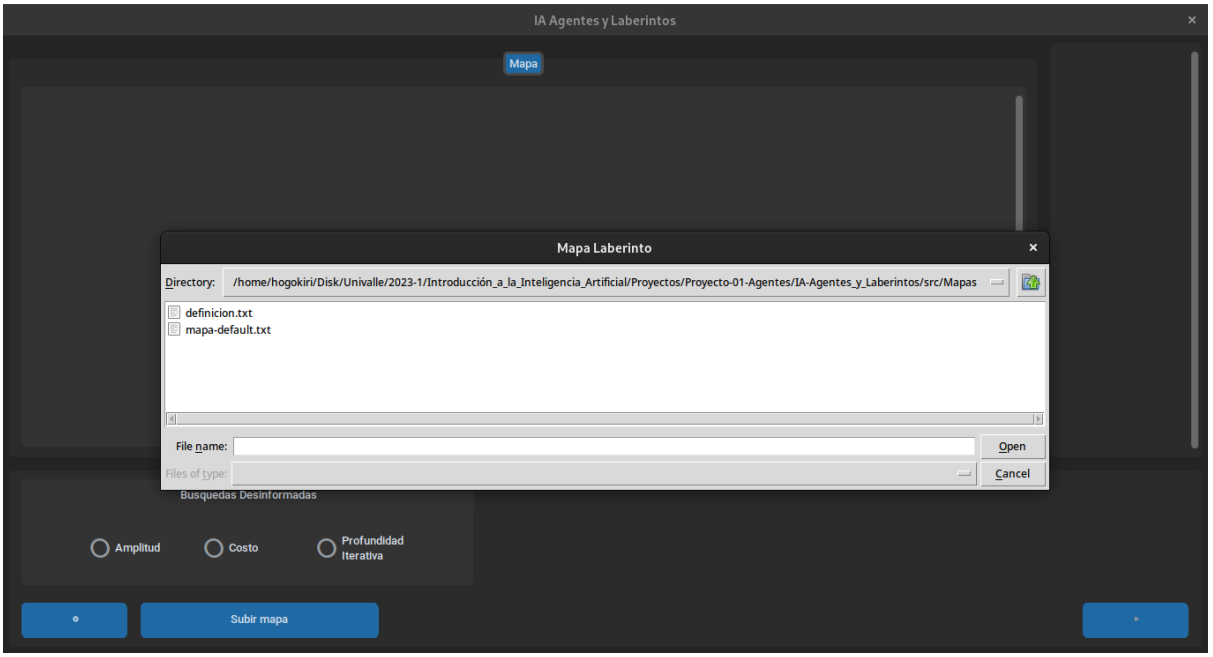
3.8. ***Modificaciones para costo:*** En el caso de costo trabajamos con una cola de prioridad de la cual extraemos la más baja.

```
heapq.heapify(cola)#ponemos la cola por prioridad siendo el primer dato costo ordenandolo por este
nodoActual = heapq.heappop(cola)#sacamos el nodo mas bajo de la cola y sus respectivas posiciones x,y
posX = nodoActual.pos.posx
posY = nodoActual.pos.posy
```

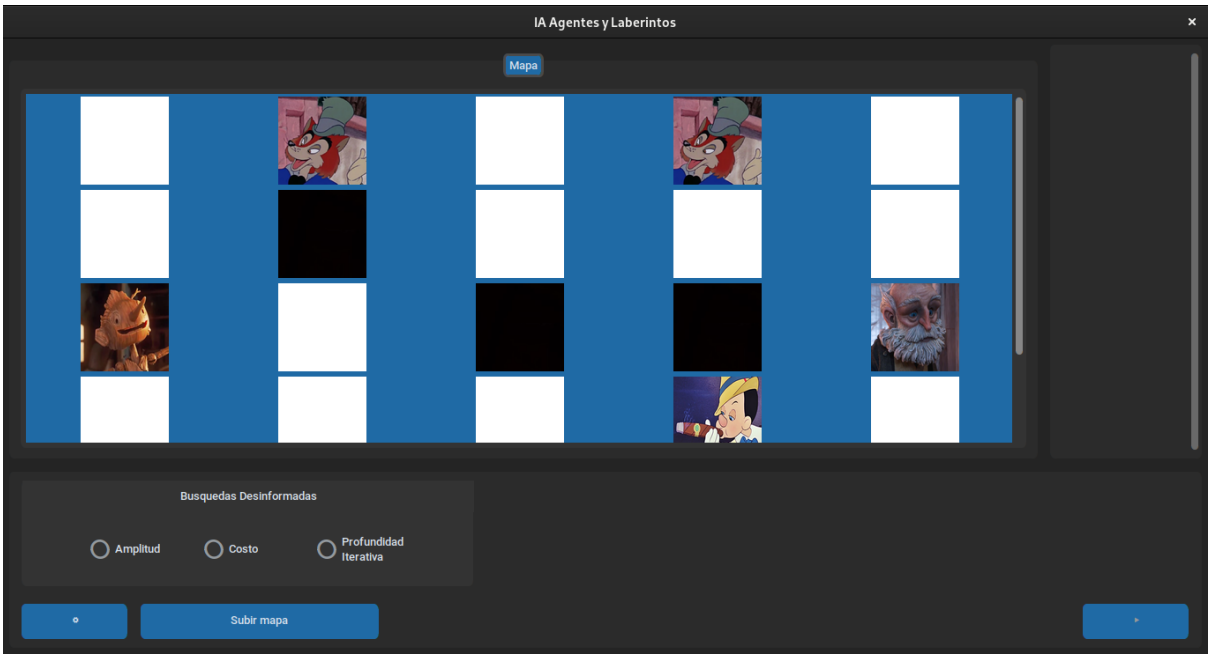
4. Instrucciones de funcionamiento y uso

4.1. Con el botón “subir mapa” se carga la matriz de un archivo de texto.



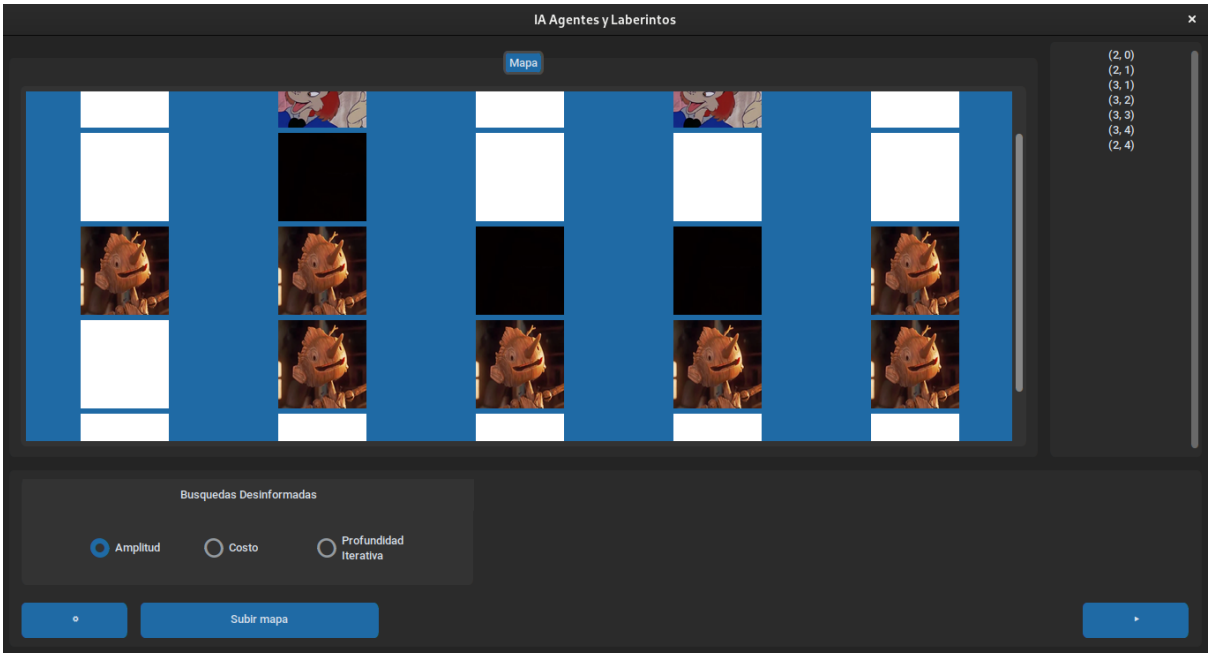

IA Agentes y Laberintos
Mapa
Mapa Laberinto
Directory:
/home/hogokiri/Disk/Univalle/2023-1/Introducción_a_la_Inteligencia_Artificial/Proyectos/Proyecto-01-Agentes/IA-Agentes_y_Laberintos/src/Mapas
definicion.txt
mapa-default.txt
File name:
Open
Files of type:
Cancel
Busquedas Desinformadas
Amplitud
Costo
Profundidad Iterativa
Subir mapa


Mapa Laberinto
Directory:
/home/hogokiri/Disk/Univalle/2023-1/Introducción_a_la_Inteligencia_Artificial/Proyectos/Proyecto-01-Agentes/IA-Agentes_y_Laberintos/src/Mapas
definicion.txt
mapa-default.txt
File name:
mapa-default.txt
Open
Files of type:
Cancel
Busquedas Desinformadas


IA Agentes y Laberintos
Mapa
Busquedas Desinformadas
Amplitud
Costo
Profundidad Iterativa
Subir mapa

4.2. Se define que tipo de búsqueda se aplicará para hacer el recorrido.

4.3. Se deja un rastro del recorrido hecho por Pinocho para llegar a Geppetto y en el lateral se muestran las coordenadas del recorrido (fila,columna).



5. Conclusión

En la mayoría de pruebas el que recorría la menor cantidad de nodos era costo, seguido de amplitud y por último profundidad iterativa (el cual recorría casi el doble de nodos). En cuanto a costo seguía el mismo orden, demostrando que por profundidad es bastante poco eficiente, no solo en cuestión de memoria si no que no da una respuesta óptima sin mencionar su costo, a pesar de eso amplitud sigue el mismo camino en ciertas situaciones siendo igual de poco óptimo, en comparación costo es bastante más óptimo en cuanto a costo aunque recorre una cantidad similar a amplitud en cuanto a nodos.

Por ende recomendamos usar el algoritmo de costo puesto que presenta un beneficio diferenciable en el recorrido.