Universidad de Guadalajara

Centro universitario de ciencias exactas e ingenierías

CUCEI

Análisis de algoritmos

Jorge Ernesto López Arce Delgado

D06

2024A

Actividad #6

Laberinto divide y vencerás

Chávez Velasco Cristian

218532484

Hernández Martínez Luis Yael

215408324

11/03/24

Introducción

Se nos presenta un laberinto representado por una matriz cuadrada (en nuestro caso de una dimensión mínima de 6x6), donde algunas celdas son caminos (representadas por 0) y otras son paredes (representadas por 1). El objetivo es encontrar un camino desde la entrada (siempre en la posición 0, 0) hasta la salida (con el valor 2) utilizando el algoritmo "divide y vencerás".

Para entender esto primero hay que hacer una breve introducción a este algoritmo, hay que entender la pregunta ¿Qué es el algoritmo divide y vencerás?

Divide y vencerás hace refencia al refrán que para resolver un problema difícil se tiene que dividir en partes mas pequeñas, tantas veces sean necesario hasta que la respuesta sea obvia, el método esta basado en la resolución recursiva de un problema como se mencionaba dividiendo el problema en dos o mas subproblemas, se continua este proceso hasta llegar a ser lo suficientemente sencillo para que se resuelva, al final las soluciones de cada uno de los subproblemas se combinan para dar con la solución final al problema original, es un buen algoritmo para casi cualquier tipo de problemas como algoritmos de ordenamiento(Quicksort), multiplicar números grandes(karatsuba), la transformada discreta de Fourier entre otros ejemplos.

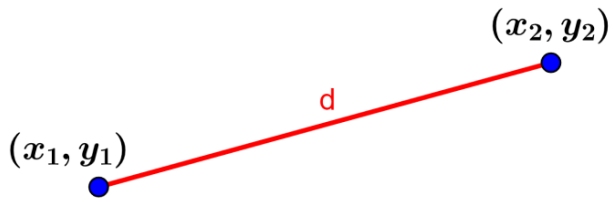
Objetivos

El objetivo principal de esta actividad es crear un laberinto con una matriz cuadrada, en la que algunas celdas son caminos y otras paredes, se tiene que implementar el algoritmo “divide y vencerás” para encontrar el camino desde la casilla de entrada, también se puede recorrer el laberinto con celdas especiales tales como teletransportes, en esta actividad al ser elaborada en equipo se dividió en tres roles los cuales tenían un objetivo propio que creo cada quien, el frontend tenía tres objetivos principales, los cuales eran elaborar la matriz, ajustar su tamaño y diseñarla correctamente en las ventanas usando tkinter, el backend tenía cuatro objetivos, tales como definir las funciones, validad las celdas y movimiento de posición del jugador, implementar el algoritmo divide y vencerás y generar la matriz con el camino resuelto, el ultimo rol que se implemento fue el de administrador del proyecto, el cual estaba atento que se subieran los avances, saber en todo momento en que fase del proyecto estaba la actividad y la elaboración del reporte y exposición.

Desarrollo:

```python
from math import sqrt


def distancia(p1,p2):
    return (sqrt(((p2[0]-p1[0])**2)+((p2[1]-p1[1])**2)))
```

Este código define una función llamada distancia que calcula la distancia euclidiana entre dos puntos en un plano bidimensional. Lo primero que se logra ver es que se importa la función sqrt (raíz cuadrada) de la librería math. La función utiliza la fórmula de distancia entre dos puntos en un plano:



$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
from tkinter import Tk, Canvas, mainloop
from tkinter import messagebox, simpledialog
import mapa as m


def mostrarMapaTkinter(mapaJuego, pasos, laberinto_resuelto=False):
    # Crear una ventana principal
    root = Tk()
    canvas = Canvas(root, width=600, height=600)
    canvas.pack()
    root.geometry("500x400")  # Establecer el tamaño de la ventana principal
```

“from tkinter import Tk, Canvas, mainloop” son de la librería de tkinter. “from tkinter import messagebox, simpledialog” importan las clases messagebox y simpledialog del módulo tkinter. Estas clases son para mostrar mensajes emergentes y diálogos simples para que el usuario interactúe con la aplicación. “import mapa as m” Esta línea llama al mapa con el alias m.

“def mostrarMapaTkinter(mapaJuego, pasos, laberinto_resuelto=False)” Esta línea define una función llamada mostrarMapaTkinter. Toma tres argumentos: mapaJuego, pasos, y laberinto_resuelto, un valor booleano que indica si el laberinto ha sido resuelto o no.

Se crea una ventana principal. También se crea un lienzo (canvas) dentro de la ventana con una anchura y altura de 600 píxeles, se establece las dimensiones de la ventana principal en 500 de ancho y 400 píxeles de alto.

```
    # Canvas para el mapa original
    canvas_original = Canvas(root, width=600, height=600)
    canvas_original.pack(side="left")
```

```
    # Canvas para el mapa con el recorrido
    canvas_recorrido = Canvas(root, width=600, height=600)
    canvas_recorrido.pack(side="right")

    # Creamos un conjunto para las coordenadas especiales
    coordenadas_especiales = set()
    coordenadas_especiales.add(tuple(mapaJuego._posicionInicial))
    coordenadas_especiales.add(tuple(mapaJuego._salida))
    for portal in mapaJuego._cordenadasPortal:
        coordenadas_especiales.add(tuple(portal))
```

Esta parte del código crea dos lienzos (canvas) dentro de la ventana principal root para representar el mapa original y el mapa con el recorrido del juego, también se crea un conjunto vacío llamado coordenadas_especiales para almacenar las coordenadas especiales del juego.

Se agregan la posición inicial del jugador al conjunto coordenadas_especiales convirtiéndola en una tupla. También la posición de salida del juego al conjunto coordenadas_especiales convirtiéndola en una tupla. Y por último se itera sobre las coordenadas de los portales del juego.

```
    # Iterar sobre cada celda del mapa
    for i in range(mapaJuego._tamanio):
        for j in range(mapaJuego._tamanio):
            x1, y1 = j * 30, i * 30
            x2, y2 = x1 + 30, y1 + 30
            # Colorear ubicaciones especiales primero
            if (i, j) in coordenadas_especiales:
                if (i, j) == tuple(mapaJuego._posicionInicial):
                    canvas.create_rectangle(x1, y1, x2, y2, fill="yellow")
                elif (i, j) == tuple(mapaJuego._salida):
                    canvas.create_rectangle(x1, y1, x2, y2, fill="green")
                elif (i, j) == tuple(mapaJuego._cordenadasPortal[0]):
                    canvas.create_rectangle(x1, y1, x2, y2, fill="blue")
                elif (i, j) == tuple(mapaJuego._cordenadasPortal[1]):
                    canvas.create_rectangle(x1, y1, x2, y2, fill="red")
                elif (i, j) == (1, 1):  # Nueva casilla "trivia"
                    canvas.create_rectangle(x1, y1, x2, y2, fill="orange")
            # Colorear el recorrido
            elif [i, j] in pasos:
                if (i, j) not in coordenadas_especiales:  # Evitar colorear
ubicaciones especiales
                    canvas.create_rectangle(x1, y1, x2, y2, fill="pink")
            # Colorear las celdas según su tipo
            elif mapaJuego._mapa[i][j] == 0:
```

```
                canvas.create_rectangle(x1, y1, x2, y2, fill="white")
            elif mapaJuego._mapa[i][j] == 1:
                canvas.create_rectangle(x1, y1, x2, y2, fill="black")
            elif mapaJuego._mapa[i][j] == 2:
                canvas.create_rectangle(x1, y1, x2, y2, fill="green")
            elif mapaJuego._mapa[i][j] == 111:  # Casilla "trivia"
                canvas.create_rectangle(x1, y1, x2, y2, fill="orange")
```

En esta parte se dibuja el mapa del juego utilizando tkinter, primero con el “for” se itera cada celda del mapa, se colorean las ubicaciones especiales, tales como salida, inicio y portales, después esta la parte que verifica que donde se vaya recorriendo se va coloreando las casillas de rosa, por último, todos los demás corresponde al tipo de celda blanco, negro o naranja para las trivias.

```
    # Mostrar mensaje si el laberinto ha sido resuelto
    if laberinto_resuelto:
        texto = "¡Laberinto resuelto! Se encontró una solución para el
laberinto."
        canvas.create_text(250, 380, text=texto, font=("Arial", 12),
fill="blue")

    root.mainloop()
```

Aquí solamente se esta escribiendo en la ventana con sus debidas características el enunciado que se imprime si se encuentra un laberinto.

```
if __name__ == "__main__":
    # Crear la matriz de mapa con un tamaño de 10x10
    mapa = m.MapaJuego(10)
    mapa.dibujarMapa()
    mostrarMapaTkinter(mapa, [])  # Pasamos una lista vacía para que al
principio no haya pasos
    resuelto = False
    pasos = []
    contador = 0
    # Intenta resolver el laberinto hasta 5 intentos
    while (not resuelto) and (contador <= 5):
        contador += 1
        try:
            pasos = mapa.resolverMapa(mapa._posicionInicial)
            resuelto = True
        except Exception as e:
            print(e)
    print(pasos)
    # Mostrar el laberinto con la solución si se encontró una
```

```
    if resuelto:
        mostrarMapaTkinter(mapa, pasos, laberinto_resuelto=True)
    else:
        messagebox.showerror("Laberinto no resuelto", "No se pudo encontrar
un camino en el laberinto.")
```

Por ultimo se crea el tamaño del mapa que en este caso es de 10x10 para que después con la función se mande a llamar, se establece una variable resuelto como falsa al principio, la lista se cera para guardar los pasos del jugador, se crea un contador en 0 para la resolución del laberinto pues con el bucle se tiene como máximo 5 intentos para que se resuelva el laberinto, si todo sale bien la variable de resuelto se muestra verdadera y se muestra el laberinto, si no se muestra un error con messagebox.

```
import numpy as np
from math import copysign as cs, ceil as ce
from random import randint as ri, shuffle as sf
from time import sleep as sl
from math import sqrt
from tkinter import messagebox, simpledialog
import utilidades as ut

class MapaJuego:
    def __init__(self, tamanio=6, posicionInicial=[0, 0]):
        self._tamanio = tamanio
        self._mapa = np.zeros((tamanio, tamanio), dtype=int)
        self._posicionInicial = posicionInicial
        self._posicionJugador = posicionInicial
        self._salida = [ri(0, tamanio - 1), tamanio - 1]
        self._cordenadasPortal = []
        self.preguntas_respuestas = {
            "¿Cuál es el río más largo del mundo?": "Amazonas",
            #etc etc
        }
```

La biblioteca “numpy” es para trabajar con matrices y arreglos multidimensionales, “math” es para redondear y calcular la raíz cuadrada, “ramdom” para generar números aleatorios, se define la clase en la que se inician los atributos del mapa como el tamaño, la matriz y las posiciones en las que se inicia, de salida coordenadas de los portales y un diccionario de preguntas y respuestas.

```
    # Comprueba si una celda dada es válida en el mapa.
    def celdaValida(self,celda):
        if (celda[0] < -1) or (celda[1] < -1):
            return False
        if (celda[0] > self._tamanio) or (celda[1] > self._tamanio):
```

```
            return False
        if (celda == self._posicionInicial)or(celda == self._salida):
            return False
        return True

    # Comprueba si una celda dada ha sido visitada.
    def celdaVisitada(self, celda):
        if (self._mapa[celda[0],celda[1]]==1):
            return True
        else:
            return False
```

Aquí son dos funciones, una es la encargada de validar si una celda es válida dentro del mapa y si no es la posición inicial ni de salida, la otra función es para verificar si ya se visito o no una celda en la cual en el valor de la celda cambia a 1 si ya fue visitada.

```
    # Muestra una pregunta trivial al jugador y verifica su respuesta.
    def mostrarTrivia(self):
        pregunta = ri(0, len(self.preguntas_respuestas) - 1)
        pregunta_texto = list(self.preguntas_respuestas.keys())[pregunta]
        respuesta_correcta = self.preguntas_respuestas[pregunta_texto]

        respuesta_jugador = simpledialog.askstring("Trivia", pregunta_texto)

        if respuesta_jugador is not None and respuesta_jugador.lower() ==
respuesta_correcta.lower():
            messagebox.showinfo("Respuesta correcta", "¡Respuesta correcta!
Puedes pasar por la puerta.")
            return True
        else:
            messagebox.showerror("Respuesta incorrecta", "Respuesta
incorrecta. Debes responder correctamente para pasar.")
            return False
```

Se elige una pregunta al azar del diccionario de preguntas, después se muestra en un cuadro de dialogo la pregunta para que pueda responderla el usuario, si la respuesta es correcta se regresa un true y un mensaje, si no un false y tienes que volver a intentar.

```
    def crearCamino(self, densidadParedes):
        densidadParedes = densidadParedes *(self._tamanio ** 2)//4
        direcciones = [(2, 0), (-2, 0), (0, 2), (0, -2)]
        for i in range(0, densidadParedes):
            x = (ri(2,self._tamanio-3)//2)*2
```

```
            y = (ri(2,self._tamanio-3)//2)*2
            if(self.celdaValida([x,y]) and (not self.celdaVisitada([x,y]))):
                self._mapa[x][y]= 1 # Marca la celda como visitada
                for dx,dy in direcciones:
                    nx = x + dx
                    ny = y + dy
                    if(self.celdaValida([nx,ny])):
                        if self._mapa[nx][ny] == 0:
                            self._mapa[nx][ny] =1
                            if(x== nx):
                                ny += int((cs(1,(ny - y))*-1))
                            if(y == ny):
                                nx += int((cs(1,(nx - x))*-1))
                            self._mapa[nx][ny] =1
        return
```

La densidad de las paredes se calcula multiplicando el tamaño del mapa al cuadrado y dividiéndolo por 4. Se define una lista de direcciones que se pueden tomar desde una celda para moverse en el mapa. Las direcciones representan movimientos de dos celdas en cualquier dirección horizontal o vertical, se itera un número de veces igual a la densidad de las paredes. En cada iteración se eligen coordenadas aleatorias dentro del mapa, si la celda es válida y no ha sido visitada, se marca como visitada asignándole el valor 1 en el mapa. Para cada dirección se calcula la siguiente celda (nx, ny) a partir de la celda actual (x, y). Si la siguiente celda es válida y está libre (tiene un valor de 0 en el mapa), se marca como parte del camino asignándole el valor 1 en el mapa. Se actualizan las coordenadas de la celda actual para mantener la continuidad. Una vez completado la función termina y no devuelve ningún valor.

```
    # Crea puertas en el mapa.
    def crearPuertas(self):
        for i in range(0, ce(self._tamanio/10)):
            puertaCreada = False
            while not puertaCreada:
                x = ri(1,self._tamanio-2)
                y = ri(1,self._tamanio-2)
                if (self._mapa[x][y] == 1):
                    if (((self._mapa[x-1][y] == 0) and  (self._mapa[x+1][y]
== 0)) or
                       ((self._mapa[x][y-1] == 0) and  (self._mapa[x][y+1]
== 0))):
                        self._mapa[x][y] = 111
                        puertaCreada = True
```

Aquí se decide cuántas puertas creare, para cada puerta se elige aleatoriamente una ubicación en el mapa que sea una pared (un área que no esté bloqueada), verifica que alrededor haya espacio vacío para colocar una puerta. Si encuentra un lugar coloca la puerta en esa ubicación y se repite este proceso hasta que todas las puertas estén colocadas en el mapa.

```
    def crearPortal(self):
        entradaCreada = False
        salidaCreada = False
        while not entradaCreada:
            x = ri(0,self._tamanio-1)
            y = ri(0,self._tamanio-1)
            if (self._mapa[x][y] == 0):
                self._mapa[x][y] = 3
                entradaCreada = True
                self._cordenadasPortal.append([x,y])

        while not salidaCreada:
            x = ri(0,self._tamanio-1)
            y = ri(0,self._tamanio-1)
            if (self._mapa[x][y] == 0):
                self._mapa[x][y] = 4
                salidaCreada = True
                self._cordenadasPortal.append([x,y])
```

Se elige aleatoriamente una posición dentro del mapa que esté vacía. Si se encuentra una posición vacía se marca como un portal de entrada (3) y se registra esa posición en la lista de coordenadas de portal. En el portal de salida pasa exactamente lo mismo.

```
    def dibujarMapa(self):
        self._mapa[self._posicionInicial[0],self._posicionInicial[1]]= 8
        self._mapa[0][2]=1
        self.crearCamino(500)
        self.crearPuertas()
        self.crearPortal()
        self._mapa[self._salida[0],self._salida[1]]= 2
        return self
```

Se marca la posición inicial del jugador en el mapa con un valor de 8, una pared en la posición (0, 2) del mapa con un valor de 1. Se utiliza el método “crearCamino” para crear un camino a través del mapa con una densidad específica de paredes, el método “crearPuertas” para colocar puertas en el mapa,

método “crearPortal” para crear un par de portales, uno de entrada y otro de salida. Por último, marca la salida del laberinto en el mapa con un valor de 2.

```
    def resolverMapa(self, posicion, pasos=[[], []]):
        menorDistancia = self._tamanio ** 2
        print(pasos)
        if posicion == self._salida:
            return pasos[0]

        # Verificar si el jugador está en una casilla de trivia
        if self._mapa[posicion[0]][posicion[1]] == 111:
            if self.mostrarTrivia():  # Mientras la respuesta sea
incorrecta, seguir mostrando la trivia
                pasos[0].append(posicion)
            else:
                return pasos[0]
```

Se inicializa “menorDistancia”con el valor del área total del laberinto. Si la posición actual coincide con la salida, se devuelve la lista de pasos tomados, si la posición es una casilla de trivia se muestra la pregunta, si la respuesta es incorrecta se devuelve la lista de pasos tomados hasta ese momento.

```
        movimientos = [(1, 0), (0, 1), (-1, 0), (0, -1)]
        sf(movimientos)
        for dx, dy in movimientos:
            nx, ny = posicion[0] + dx, posicion[1] + dy
            if (not ((nx < 0) or (ny < 0) or (nx >= self._tamanio) or (ny >=
self._tamanio))) and (
                    not ([nx, ny] in pasos[0]) and not ([nx, ny] in
pasos[1])):
                if ((self._mapa[nx][ny] == 0) or (self._mapa[nx][ny] == 111)
or ((self._mapa[nx][ny] == 2))):
                    if (ut.distancia([nx, ny], self._salida) <=
menorDistancia):
                        siguientePaso = [nx, ny]
                        menorDistancia = ut.distancia([nx, ny],
self._salida)

                elif (self._mapa[nx][ny] == 3):
                    if (ut.distancia([nx, ny], self._salida) <=
menorDistancia):
                        pasos[0].append([nx, ny])
                        nx, ny = self._cordenadasPortal[1][0],
self._cordenadasPortal[1][1]
                        siguientePaso = [nx, ny]
```

```
                        menorDistancia = ut.distancia([nx, ny],
self._salida)

                elif (self._mapa[nx][ny] == 4):
                    if (ut.distancia([nx, ny], self._salida) <=
menorDistancia):
                        pasos[0].append([nx, ny])
                        nx, ny = nx, ny = self._cordenadasPortal[0][0],
self._cordenadasPortal[0][1]
                        siguientePaso = [nx, ny]
                        menorDistancia = ut.distancia([nx, ny],
self._salida)

        try:
            pasos[0].append(siguientePaso) # Añade el siguiente paso a la
lista de pasos
            self.resolverMapa(siguientePaso, pasos)
        except Exception as er:
            pasos[1].append(pasos[0].pop())
            self.resolverMapa(pasos[0][-2], pasos)

        return pasos[0]
```

Se definen los posibles movimientos como desplazamientos en las direcciones: arriba, abajo, izquierda y derecha. Los movimientos se hacen aleatoriamente para evitar patrones predecibles, para cada dirección en los movimientos se calcula la nueva posición nx, ny.

Se verifica si la nueva posición está dentro de los límites del laberinto y si no ha sido visitada previamente. Si la celda está vacía, es una casilla de trivia, o es la salida, se considera como un posible siguiente paso hacia la salida si esta distancia es menor que la menor distancia conocida. Si la celda contiene un portal de entrada (3), se añade la posición actual a los pasos tomados y se mueve al jugador al portal de salida. Si la celda contiene un portal de salida (4), se añade la posición actual a los pasos tomados y se mueve al jugador al portal de entrada.

Se intenta resolver el laberinto desde la siguiente posición encontrada. Si se produce un error durante la resolución del laberinto, se retrocede un paso y se intenta otra dirección. Al final se devuelve la lista de pasos tomados hasta el momento.

Código:

Main:

```
from tkinter import Tk, Canvas, mainloop
```

```python
from tkinter import messagebox, simpledialog
import mapa as m


def mostrarMapaTkinter(mapaJuego, pasos, laberinto_resuelto=False):
    # Crear una ventana principal
    root = Tk()
    canvas = Canvas(root, width=600, height=600)
    canvas.pack()
    root.geometry("500x400")  # Establecer el tamaño de la ventana principal

    # Canvas para el mapa original
    canvas_original = Canvas(root, width=600, height=600)
    canvas_original.pack(side="left")

    # Canvas para el mapa con el recorrido
    canvas_recorrido = Canvas(root, width=600, height=600)
    canvas_recorrido.pack(side="right")

    # Creamos un conjunto para las coordenadas especiales
    coordenadas_especiales = set()
    coordenadas_especiales.add(tuple(mapaJuego._posicionInicial))
    coordenadas_especiales.add(tuple(mapaJuego._salida))
    for portal in mapaJuego._cordenadasPortal:
        coordenadas_especiales.add(tuple(portal))

    # Iterar sobre cada celda del mapa
    for i in range(mapaJuego._tamanio):
        for j in range(mapaJuego._tamanio):
            x1, y1 = j * 30, i * 30
            x2, y2 = x1 + 30, y1 + 30
            # Colorear ubicaciones especiales primero
            if (i, j) in coordenadas_especiales:
                if (i, j) == tuple(mapaJuego._posicionInicial):
                    canvas.create_rectangle(x1, y1, x2, y2, fill="yellow")
                elif (i, j) == tuple(mapaJuego._salida):
                    canvas.create_rectangle(x1, y1, x2, y2, fill="green")
                elif (i, j) == tuple(mapaJuego._cordenadasPortal[0]):
                    canvas.create_rectangle(x1, y1, x2, y2, fill="blue")
                elif (i, j) == tuple(mapaJuego._cordenadasPortal[1]):
                    canvas.create_rectangle(x1, y1, x2, y2, fill="red")
                elif (i, j) == (1, 1):  # Nueva casilla "trivia"
                    canvas.create_rectangle(x1, y1, x2, y2, fill="orange")
            # Colorear el recorrido
            elif [i, j] in pasos:
```

```
                if (i, j) not in coordenadas_especiales:  # Evitar colorear
ubicaciones especiales
                    canvas.create_rectangle(x1, y1, x2, y2, fill="pink")
            # Colorear las celdas según su tipo
            elif mapaJuego._mapa[i][j] == 0:
                canvas.create_rectangle(x1, y1, x2, y2, fill="white")
            elif mapaJuego._mapa[i][j] == 1:
                canvas.create_rectangle(x1, y1, x2, y2, fill="black")
            elif mapaJuego._mapa[i][j] == 2:
                canvas.create_rectangle(x1, y1, x2, y2, fill="green")
            elif mapaJuego._mapa[i][j] == 111:  # Casilla "trivia"
                canvas.create_rectangle(x1, y1, x2, y2, fill="orange")

    # Mostrar mensaje si el laberinto ha sido resuelto
    if laberinto_resuelto:
        texto = "¡Laberinto resuelto! Se encontró una solución para el
laberinto."
        canvas.create_text(250, 380, text=texto, font=("Arial", 12),
fill="blue")

    root.mainloop()

if __name__ == "__main__":
    # Crear la matriz de mapa con un tamaño de 10x10
    mapa = m.MapaJuego(10)
    mapa.dibujarMapa()
    mostrarMapaTkinter(mapa, [])  # Pasamos una lista vacía para que al
principio no haya pasos
    resuelto = False
    pasos = []
    contador = 0
    # Intenta resolver el laberinto hasta 5 intentos
    while (not resuelto) and (contador <= 5):
        contador += 1
        try:
            pasos = mapa.resolverMapa(mapa._posicionInicial)
            resuelto = True
        except Exception as e:
            print(e)
    print(pasos)
    # Mostrar el laberinto con la solución si se encontró una
    if resuelto:
        mostrarMapaTkinter(mapa, pasos, laberinto_resuelto=True)
    else:
```

```
        messagebox.showerror("Laberinto no resuelto", "No se pudo encontrar
un camino en el laberinto.")
```

Mapa:

```
import numpy as np
from math import copysign as cs, ceil as ce
from random import randint as ri, shuffle as sf
from time import sleep as sl
from math import sqrt
from tkinter import messagebox, simpledialog
import utilidades as ut

class MapaJuego:
    def __init__(self, tamanio=6, posicionInicial=[0, 0]):
        self._tamanio = tamanio
        self._mapa = np.zeros((tamanio, tamanio), dtype=int)
        self._posicionInicial = posicionInicial
        self._posicionJugador = posicionInicial
        self._salida = [ri(0, tamanio - 1), tamanio - 1]
        self._cordenadasPortal = []
        self.preguntas_respuestas = {
            "¿Cuál es el río más largo del mundo?": "Amazonas",
            #etc etc
        }

    # Comprueba si una celda dada es válida en el mapa.
    def celdaValida(self,celda):
        if (celda[0] < -1) or (celda[1] < -1):
            return False
        if (celda[0] > self._tamanio) or (celda[1] > self._tamanio):
            return False
        if (celda == self._posicionInicial)or(celda == self._salida):
            return False
        return True

    # Comprueba si una celda dada ha sido visitada.
    def celdaVisitada(self, celda):
        if (self._mapa[celda[0],celda[1]]==1):
            return True
        else:
            return False

    # Muestra una pregunta trivial al jugador y verifica su respuesta.
    def mostrarTrivia(self):
```

```
        pregunta = ri(0, len(self.preguntas_respuestas) - 1)
        pregunta_texto = list(self.preguntas_respuestas.keys())[pregunta]
        respuesta_correcta = self.preguntas_respuestas[pregunta_texto]

        respuesta_jugador = simpledialog.askstring("Trivia", pregunta_texto)

        if respuesta_jugador is not None and respuesta_jugador.lower() ==
respuesta_correcta.lower():
            messagebox.showinfo("Respuesta correcta", "¡Respuesta correcta!
Puedes pasar por la puerta.")
            return True
        else:
            messagebox.showerror("Respuesta incorrecta", "Respuesta
incorrecta. Debes responder correctamente para pasar.")
            return False

    def crearCamino(self, densidadParedes):
        densidadParedes = densidadParedes *(self._tamanio ** 2)//4
        direcciones = [(2, 0), (-2, 0), (0, 2), (0, -2)]
        for i in range(0, densidadParedes):
            x = (ri(2,self._tamanio-3)//2)*2
            y = (ri(2,self._tamanio-3)//2)*2
            if(self.celdaValida([x,y]) and (not self.celdaVisitada([x,y]))):
                self._mapa[x][y]= 1 # Marca la celda como visitada
                for dx,dy in direcciones:
                    nx = x + dx
                    ny = y + dy
                    if(self.celdaValida([nx,ny])):
                        if self._mapa[nx][ny] == 0:
                            self._mapa[nx][ny] =1
                            if(x== nx):
                                ny += int((cs(1,(ny - y))*-1))
                            if(y == ny):
                                nx += int((cs(1,(nx - x))*-1))
                            self._mapa[nx][ny] =1
        return


    # Crea puertas en el mapa.
    def crearPuertas(self):
        for i in range(0, ce(self._tamanio/10)):
            puertaCreada = False
            while not puertaCreada:
                x = ri(1,self._tamanio-2)
                y = ri(1,self._tamanio-2)
```

```
                if (self._mapa[x][y] == 1):
                    if (((self._mapa[x-1][y] == 0) and  (self._mapa[x+1][y]
== 0)) or
                       ((self._mapa[x][y-1] == 0) and  (self._mapa[x][y+1]
== 0))):
                        self._mapa[x][y] = 111
                        puertaCreada = True

    def crearPortal(self):
        entradaCreada = False
        salidaCreada = False
        while not entradaCreada:
            x = ri(0,self._tamanio-1)
            y = ri(0,self._tamanio-1)
            if (self._mapa[x][y] == 0):
                self._mapa[x][y] = 3
                entradaCreada = True
                self._cordenadasPortal.append([x,y])

        while not salidaCreada:
            x = ri(0,self._tamanio-1)
            y = ri(0,self._tamanio-1)
            if (self._mapa[x][y] == 0):
                self._mapa[x][y] = 4
                salidaCreada = True
                self._cordenadasPortal.append([x,y])

    def dibujarMapa(self):
        self._mapa[self._posicionInicial[0],self._posicionInicial[1]]= 8
        self._mapa[0][2]=1
        self.crearCamino(500)
        self.crearPuertas()
        self.crearPortal()
        self._mapa[self._salida[0],self._salida[1]]= 2
        return self


    def resolverMapa(self, posicion, pasos=[[], []]):
        menorDistancia = self._tamanio ** 2
        print(pasos)
        if posicion == self._salida:
            return pasos[0]

        # Verificar si el jugador está en una casilla de trivia
        if self._mapa[posicion[0]][posicion[1]] == 111:
```

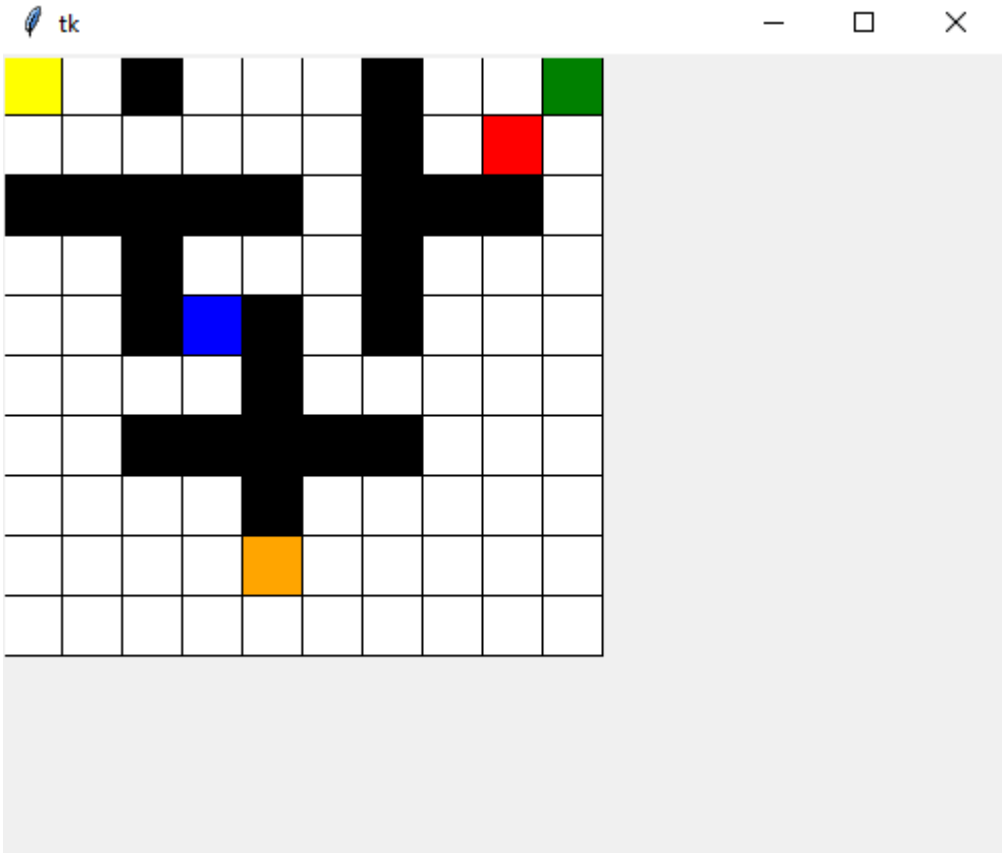
```
            if self.mostrarTrivia():  # Mientras la respuesta sea
incorrecta, seguir mostrando la trivia
                pasos[0].append(posicion)
            else:
                return pasos[0]

        movimientos = [(1, 0), (0, 1), (-1, 0), (0, -1)]
        sf(movimientos)
        for dx, dy in movimientos:
            nx, ny = posicion[0] + dx, posicion[1] + dy
            if (not ((nx < 0) or (ny < 0) or (nx >= self._tamanio) or (ny >=
self._tamanio))) and (
                    not ([nx, ny] in pasos[0]) and not ([nx, ny] in
pasos[1])):
                if ((self._mapa[nx][ny] == 0) or (self._mapa[nx][ny] == 111)
or ((self._mapa[nx][ny] == 2))):
                    if (ut.distancia([nx, ny], self._salida) <=
menorDistancia):
                        siguientePaso = [nx, ny]
                        menorDistancia = ut.distancia([nx, ny],
self._salida)

                elif (self._mapa[nx][ny] == 3):
                    if (ut.distancia([nx, ny], self._salida) <=
menorDistancia):
                        pasos[0].append([nx, ny])
                        nx, ny = self._cordenadasPortal[1][0],
self._cordenadasPortal[1][1]
                        siguientePaso = [nx, ny]
                        menorDistancia = ut.distancia([nx, ny],
self._salida)

                elif (self._mapa[nx][ny] == 4):
                    if (ut.distancia([nx, ny], self._salida) <=
menorDistancia):
                        pasos[0].append([nx, ny])
                        nx, ny = nx, ny = self._cordenadasPortal[0][0],
self._cordenadasPortal[0][1]
                        siguientePaso = [nx, ny]
                        menorDistancia = ut.distancia([nx, ny],
self._salida)

        try:
            pasos[0].append(siguientePaso) # Añade el siguiente paso a la
lista de pasos
```

```
            self.resolverMapa(siguientePaso, pasos)
        except Exception as er:
            pasos[1].append(pasos[0].pop())
            self.resolverMapa(pasos[0][-2], pasos)

        return pasos[0]
```

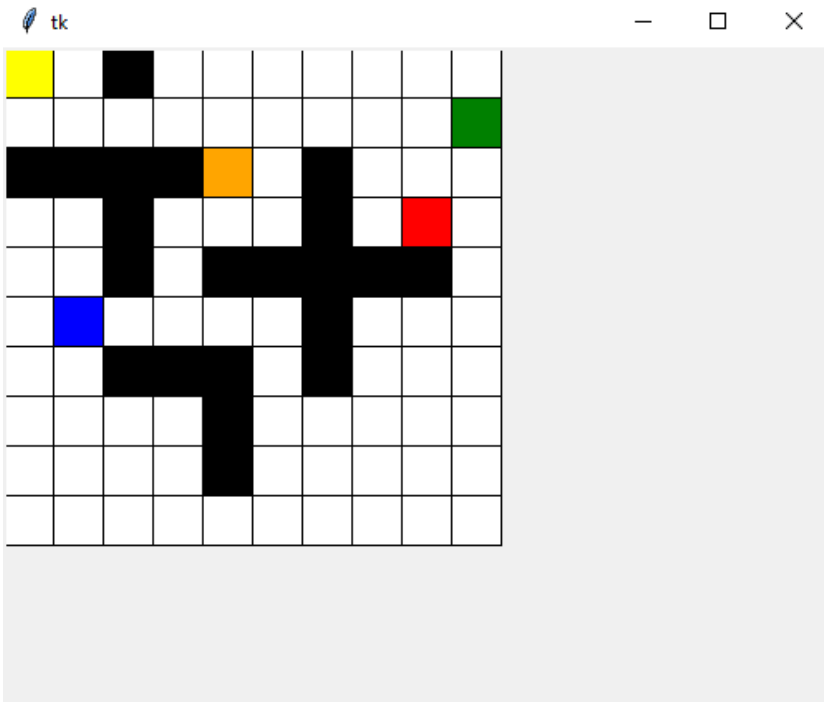
Utilidades:

```
from math import sqrt


def distancia(p1,p2):
    return (sqrt(((p2[0]-p1[0])**2)+((p2[1]-p1[1])**2)))
```

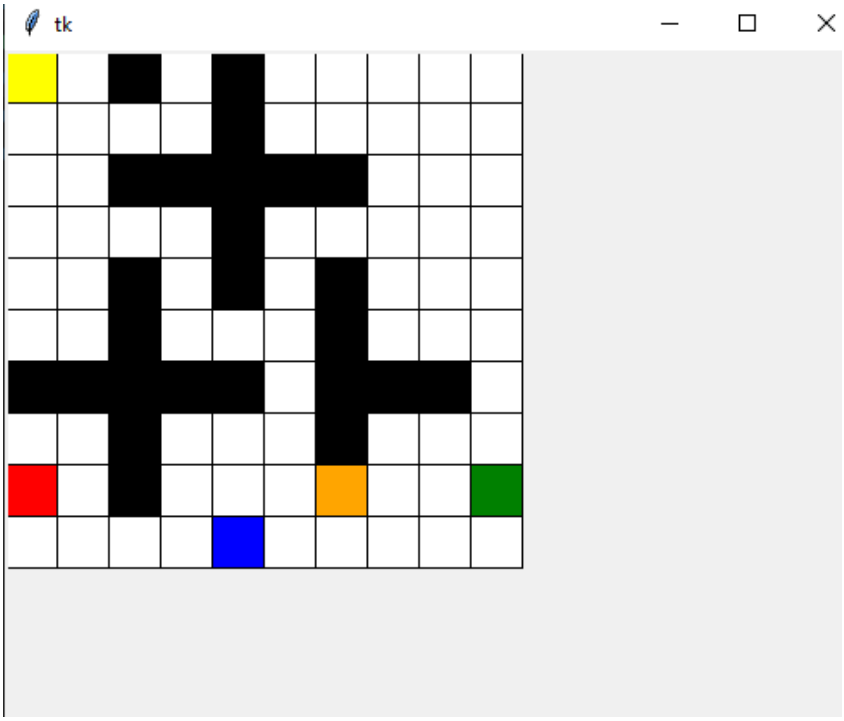
Capturas del funcionamiento del código:

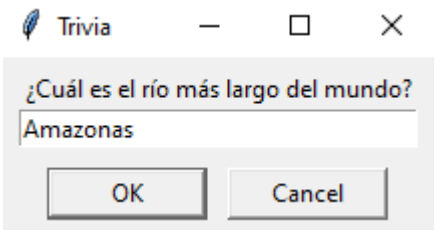


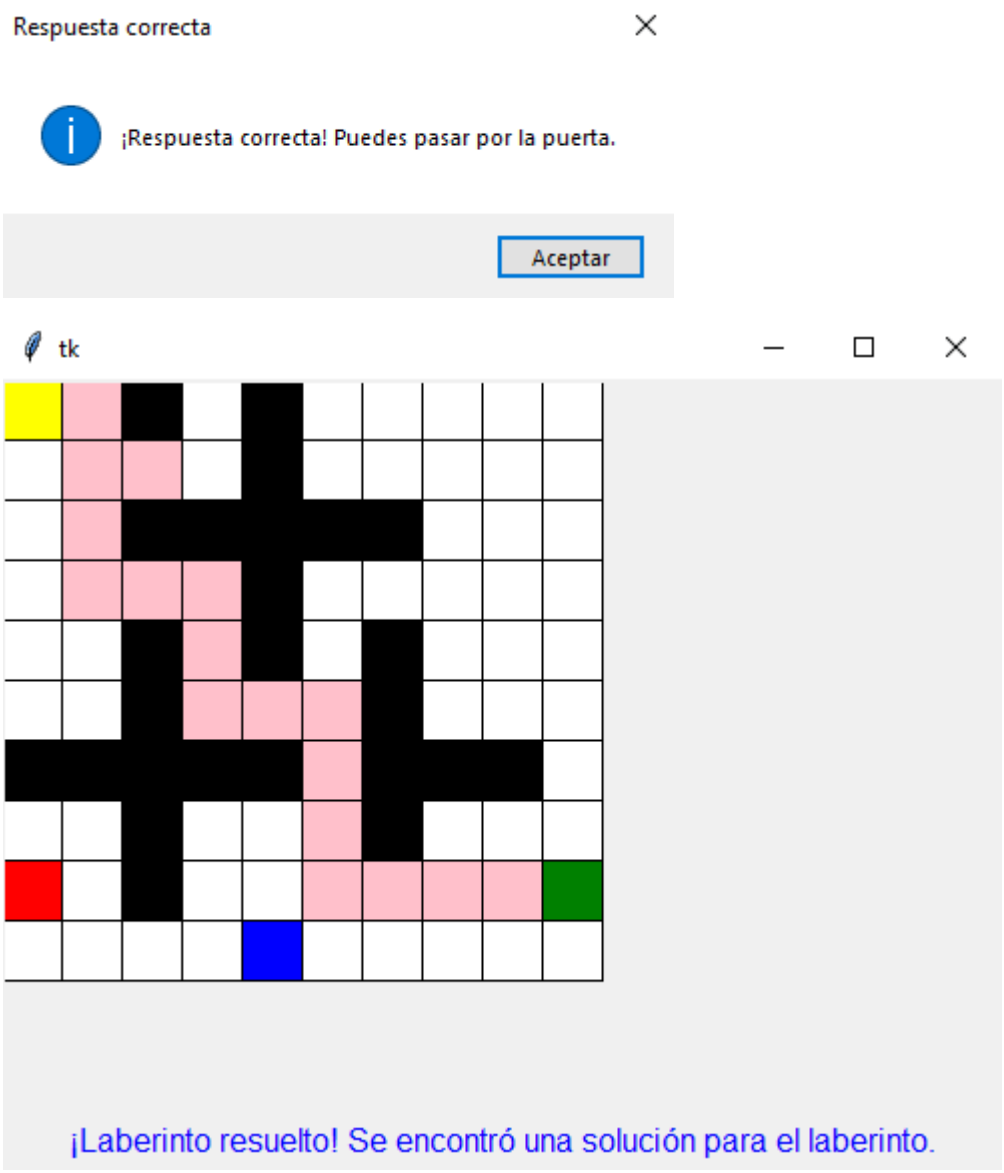

tk
¡Laberinto resuelto! Se encontró una solución para el laberinto.


tk


tk
¡Laberinto resuelto! Se encontró una solución para el laberinto.


tk


Trivia
¿Cuál es el río más largo del mundo?
Amazonas
OK
Cancel



Conclusiones Cristian:

En lo personal al tener el puesto del administrador pude saber de primera mano que no es tan fácil como parece, pues es quien básicamente da la cara por el equipo en todo momento, tiene que lidiar con los problemas que se presenten, este supervisando a los demás y al final creo que es algo estresante en algunas ocasiones, quizás si nos hace falta tal como lo menciono el profe el desarrollo de estas cualidades blandas para en el futuro poder ser mucho mejores, tanto profesionalmente como personalmente.

Conclusiones Luis:

El análisis del problema me llevo a tener que entender o buscar primero una manera de crear un laberinto que fuera posible resolver de manera aleatoria en cada vez que se ejecutara el programa que en sí mismo llevo un reto y el segundo

reto ya fue entender el problema de la resolución del laberinto basándome en que tengo un punto de salida y siempre una posible solución ir dividiendo el problema hasta su parte más fundamental que fue el moverse a una casilla y de esta manera junto con saber que casillas ya había visitado y ver que movimiento sería el más óptimo para llegar a la salida fue como hice posible la resolución del problema.

Referencias

Giraldo Acosta, M. (2016). Ambiente interactivo para la visualización de conceptos asociados a las técnicas de diseño de algoritmos.