UNIVERSIDAD POLITÉCNICA DE TLAXCALA

"REGIÓN PONIENTE"

REPORTE CAMINOS EN UNA CUADRICULA

SISTEMAS INTELIGENTES

PROFA: ING. VANESA TENOPALA ZAVALA

INTREGRANTES DE EQUIPO:

INGRID MAGALI CRUZ MALDONADO
ANTONIO OLVERA ESPINOZA
DIANA LAURA PEÑA ROJAS
RUBI JUAREZ GALLARDO
LEO XELHUATZI LIRA

INGENIERIA: SISTEMAS COMPUTACIONALES

GRADO: 8to

GRUPO: "A"

ENERO – ABRIL 2024

RESUMEN

Ambos programas simulan la búsqueda de caminos desde una posición inicial hasta una posición objetivo en un laberinto. El primero utiliza la biblioteca Pygame para la representación gráfica y encuentra todas las rutas posibles, mientras que el segundo utiliza Tkinter y encuentra todos los caminos posibles evitando el recuadro en el centro.

INTRODUCCIÓN

Estos programas resuelven el problema de encontrar caminos en un laberinto desde una posición inicial hasta una posición objetivo. Ambos emplean el análisis para explorar las posibles rutas y evitan obstáculos específicos.

DESARROLLO

Programa 1 (Pygame): Este programa tiene como objetivo principal visualizar gráficamente todas las posibles rutas desde la posición inicial hasta la posición objetivo en un laberinto. Utiliza la biblioteca Pygame para crear una representación gráfica interactiva del laberinto, donde el ratón busca el queso evitando obstáculos, y muestra todas las rutas exploradas.

```
import pygame
import sys


laberinto_size = 5
cell_size = 50
laberinto = [[0] * laberinto_size for _ in range(laberinto_size)]


WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)


def draw(laberinto, raton_pos, queso_pos, ruta_actual=None,
contador_caminos=0):
    screen.fill(WHITE)


    for i in range(laberinto_size):
        for j in range(laberinto_size):
            pygame.draw.rect(screen, BLACK, ((laberinto_size - 1 - j) *
cell_size, i * cell_size, cell_size, cell_size), 1)
```

```
    pygame.draw.circle(screen, RED, (raton_pos[1] * cell_size + cell_size //
2, (laberinto_size - 1 - raton_pos[0]) * cell_size + cell_size // 2),
cell_size // 3)
    pygame.draw.circle(screen, RED, (queso_pos[1] * cell_size + cell_size //
2, (laberinto_size - 1 - queso_pos[0]) * cell_size + cell_size // 2),
cell_size // 3)


    if ruta_actual:
        for pos in ruta_actual:
            pygame.draw.rect(screen, RED, ((laberinto_size - 1 - pos[1]) *
cell_size, pos[0] * cell_size, cell_size, cell_size))


    font = pygame.font.Font(None, 36)
    text = font.render(f"Caminos: {contador_caminos}", True, BLACK)
    screen.blit(text, (10, 10))

    pygame.display.flip()


pygame.init()


window_size = (laberinto_size * cell_size, laberinto_size * cell_size)
screen = pygame.display.set_mode(window_size)
pygame.display.set_caption("Laberinto del Ratón")


raton_pos = (laberinto_size - 1, laberinto_size - 1)
queso_pos = (0, 0)


rutas_posibles = []

def encontrar_rutas(laberinto, fila, columna, ruta_actual):
    if fila == columna == 0:
        rutas_posibles.append(ruta_actual[:])
        return

    if fila - 1 >= 0 and (fila - 1, columna) not in ruta_actual:
        ruta_actual.append((fila - 1, columna))
        encontrar_rutas(laberinto, fila - 1, columna, ruta_actual)
        ruta_actual.pop()

    if columna - 1 >= 0 and (fila, columna - 1) not in ruta_actual:
        ruta_actual.append((fila, columna - 1))
        encontrar_rutas(laberinto, fila, columna - 1, ruta_actual)
        ruta_actual.pop()
```

```
encontrar_rutas(laberinto, laberinto_size - 1, laberinto_size - 1,
[(laberinto_size - 1, laberinto_size - 1)])

for index, ruta in enumerate(rutas_posibles):
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    draw(laberinto, raton_pos, queso_pos, ruta, contador_caminos=index + 1)
    pygame.time.delay(1000)
    pygame.event.clear()


pygame.quit()
sys.exit()
```

EJECUCIÓN


Laberinto del ...
Caminos: 24


Laberinto del ...
Caminos: 57


Laberinto del ...
Caminos: 5

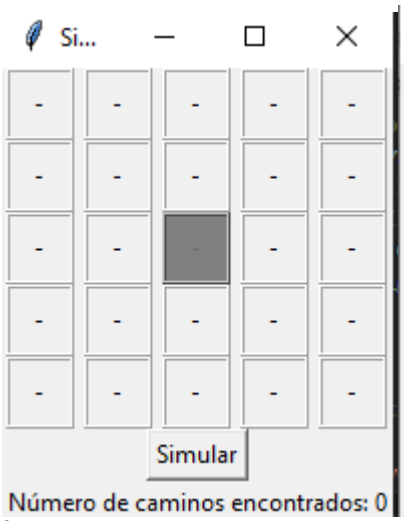

Laberinto del ...
Caminos: 29

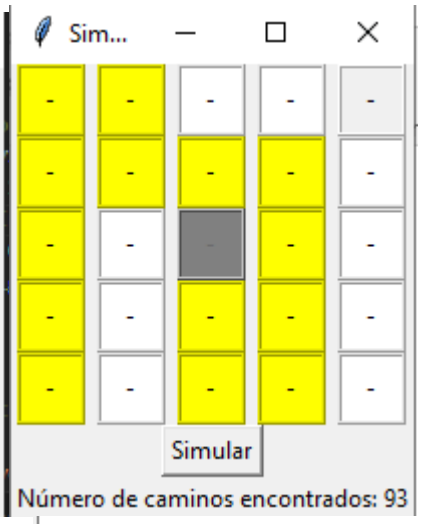
Programa 2 (Tkinter): La funcionalidad principal de este programa es simular la búsqueda de caminos desde una posición inicial hasta una posición objetivo en un cuadrado de dimensiones predefinidas. Utilizando la interfaz gráfica de Tkinter, el programa encuentra y muestra todos los caminos posibles, evitando el recuadro en el centro del cuadrado, y actualiza en tiempo real el número de caminos encontrados.

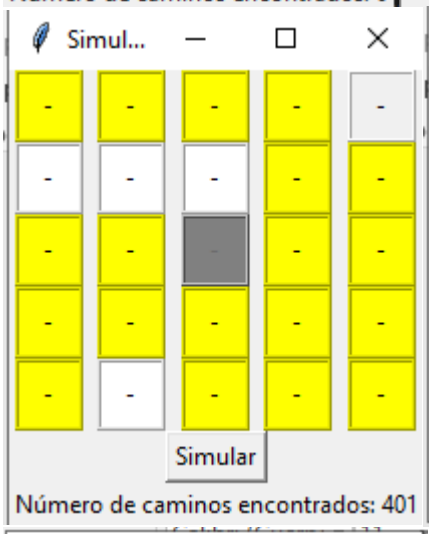
```
import tkinter as tk
import time

# Función para encontrar todos los caminos posibles
def encontrar_caminos(x, y, visitado, contador, camino_actual):
    if x == 0 and y == 4:  # Si se alcanza la posición del queso
        contador[0] += 1  # Incrementar el contador de caminos
        camino_actual = [(4, 0)]  # Reiniciar el camino
        resultado_label.config(text=f"Número de caminos encontrados:
{contador[0]}")  # Actualizar el conteo en tiempo real
        root.update()
        return

    # Marcamos la casilla actual como visitada
    visitado[x][y] = True
    cuadro[x][y].config(bg="yellow")
    root.update()

    # Revisamos las cuatro direcciones posibles (arriba, abajo, izquierda,
derecha)
    direcciones = [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]
    for dx, dy in direcciones:
        if (0 <= dx < 5 and 0 <= dy < 5 and not visitado[dx][dy] and
                not (dx == 2 and dy == 2)):  # No permitir pasar por el cuadro
bloqueado
            time.sleep(0.0)  # Ralentizar el movimiento del ratón
            visitado[dx][dy] = True
            camino_actual.append((dx, dy))
            encontrar_caminos(dx, dy, visitado, contador, camino_actual)
            camino_actual.pop()
            visitado[dx][dy] = False

    # Desmarcamos la casilla actual como no visitada para retroceder
    visitado[x][y] = False
    cuadro[x][y].config(bg="white")
    root.update()

# Función para iniciar la simulación
def simular():
    contador = [0]  # Contador de caminos
    visitado = [[False]*5 for _ in range(5)]  # Matriz de visitados
    camino_actual = [(4, 0)]  # Camino actual del ratón
```

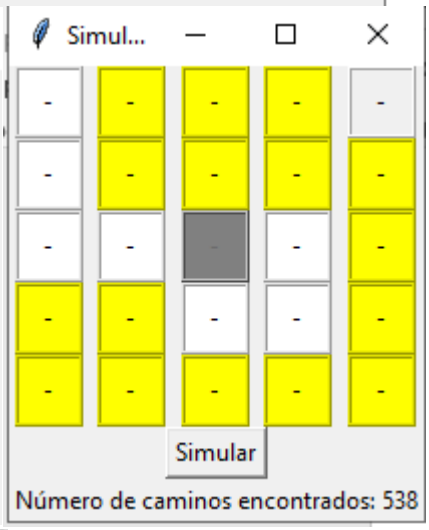
```
    encontrar_caminos(4, 0, visitado, contador, camino_actual)

# Crear la ventana principal
root = tk.Tk()
root.title("Simulación de búsqueda de caminos")

# Crear matriz para representar el cuadro
cuadro = [[None]*5 for _ in range(5)]

# Crear el cuadro
for i in range(5):
    for j in range(5):
        cuadro[i][j] = tk.Label(root, text="-", width=4, height=2,
relief="ridge", borderwidth=2)
        cuadro[i][j].grid(row=i, column=j)

# Bloquear el cuadro central
cuadro[2][2].config(bg="gray", state="disabled")

# Botón para iniciar la simulación
simular_button = tk.Button(root, text="Simular", command=simular)
simular_button.grid(row=5, column=0, columnspan=5)

# Etiqueta para mostrar el resultado
resultado_label = tk.Label(root, text="Número de caminos encontrados: 0")
resultado_label.grid(row=6, column=0, columnspan=5)

# Ejecutar la ventana principal
root.mainloop()
```

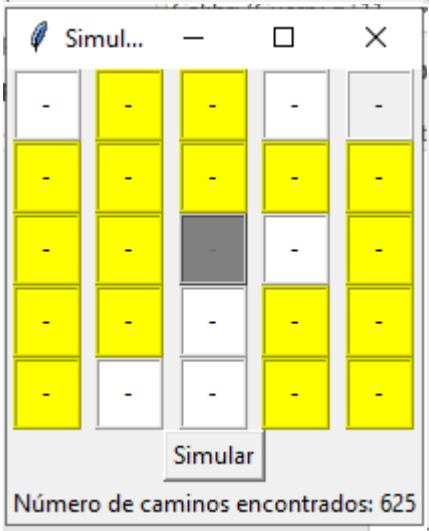
EJECUCIÓN

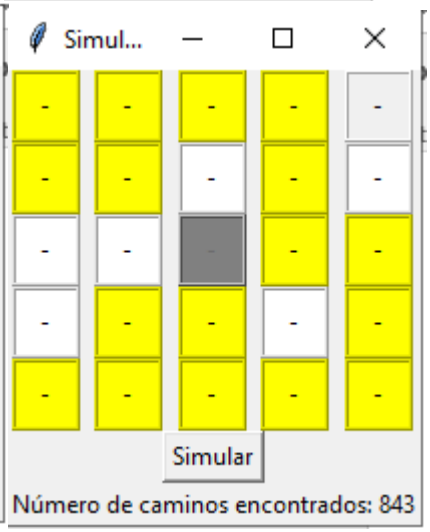

Si...
Simular
Número de caminos encontrados: 0


Sim...
Simular
Número de caminos encontrados: 93


Simul...
Simular
Número de caminos encontrados: 401


Simul...
Simular
Número de caminos encontrados: 538


Simul...
Simular
Número de caminos encontrados: 625


Simul...
Simular
Número de caminos encontrados: 843

CONCLUSIÓN

Ambos programas han sido diseñados de manera eficiente, proporcionando una experiencia interactiva y visual clara al ejecutarlos. En el caso del programa utilizando Pygame, la interfaz gráfica permite una representación visual intuitiva de las múltiples rutas en el laberinto, facilitando la comprensión de la exploración realizada por el ratón en busca del queso.

Por otro lado, el programa con Tkinter presenta una interfaz que muestra en tiempo real la simulación de búsqueda de caminos, proporcionando una visión detallada del proceso. La rápida visualización de los resultados obtenidos durante la ejecución refleja el éxito de la implementación, cumpliendo con el objetivo de encontrar y mostrar de manera efectiva en todos los caminos posibles, evitando obstáculos definidos.

Ambas implementaciones han logrado sus respectivos objetivos al proporcionar representaciones visuales claras y permitir una rápida comprensión de los resultados obtenidos.



S I C

UPTrep
UNIVERSIDAD POLITECNICA
DE TLAXCALA
REGION PONIENTE