#### PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

#### FACULTAD DE INGENIERÍA

#### ESCUELA DE INGENIERÍA INFORMÁTICA

Tarea

Sistemas Operativos

###### Anaís Monserrat Foix Monardes 20.834.761-6

###### Francisco Andrés Muñoz Alarcón 20.242.456-2

###### Asignatura: INF2341-1

###### Profesor: Iván Mercado Bermúdez

###### Carrera: Ingeniería de Ejecución Informática

###### Noviembre 2021

**Índice**

[Introducción III](#_Toc87205330)

[Solución del Ejercicio IV](#_Toc87205331)

[*Declaración de variables e inicio de programa* IV](#_Toc87205332)

[*Solución Hilos usando semáforos y Comparativa recorrido* IV](#_Toc87205333)

[*Dibujo programa* V](#_Toc87205335)

[Código documentado VI](#_Toc87205336)

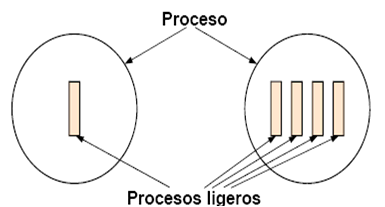
[Resultados IX](#_Toc87205337)

[Conclusión X](#_Toc87205338)

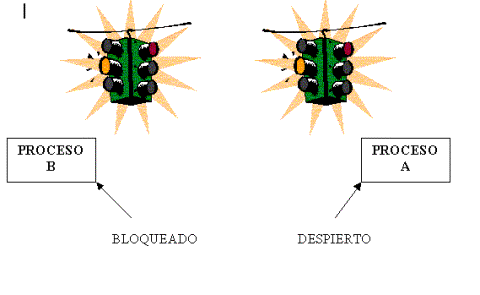
# Introducción

En este informe se explicará la manera de abordar un ejercicio presentado y cómo se soluciona la problemática mostrando detalladamente el proceso de desarrollo y la creación del programa basado en el lenguaje Python y herramientas vistas en este curso tales como semáforos e hilos.

Los hilos son bloques de códigos que pueden ser ejecutados como programas independientes, esto nos permite ejecutar varios bloques de manera simultánea.



Por su parte los semáforos son un mecanismo de sincronización de procesos, permiten asistir al planificador del sistema operativo.



El ejercicio abordado trata sobre la resolución de un laberinto, el cual debe recorrer usando múltiples hebras todo el laberinto, y con ello encontrar la posible escapatoria de este. Para lograrlo se debe llenar una matriz e ir desplazando a nuestro protagonista por esta, hasta encontrar la salida.

# Solución del Ejercicio

Uno de los primeros pasos a realizar es descargar e instalar la herramienta de Python, para luego empezar a desarrollar el código a ejecutar. También es necesario tener instalado un compilador, que en nuestro caso usaremos “Visual Studio Code”.

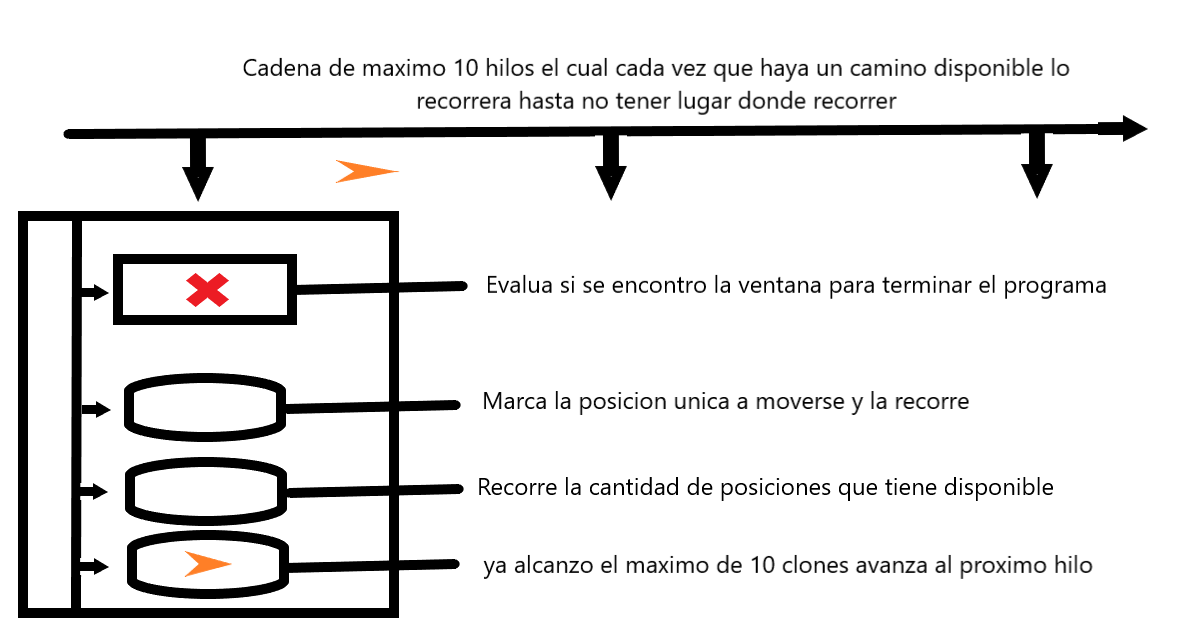
## *Declaración de variables e inicio de programa*

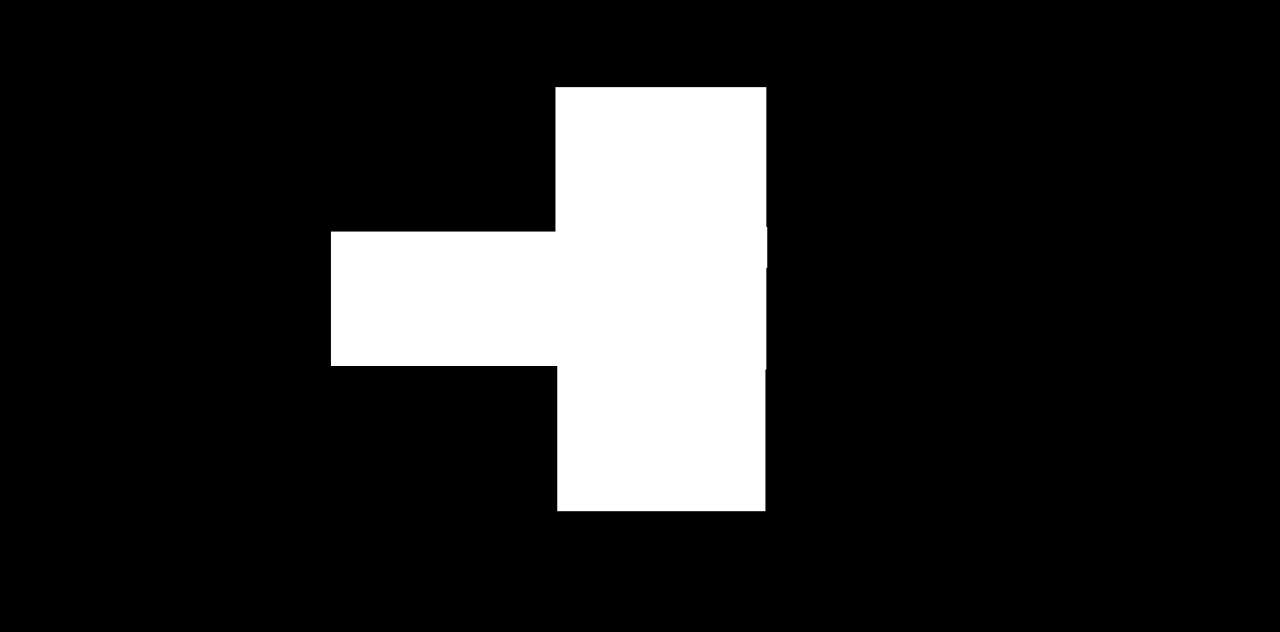
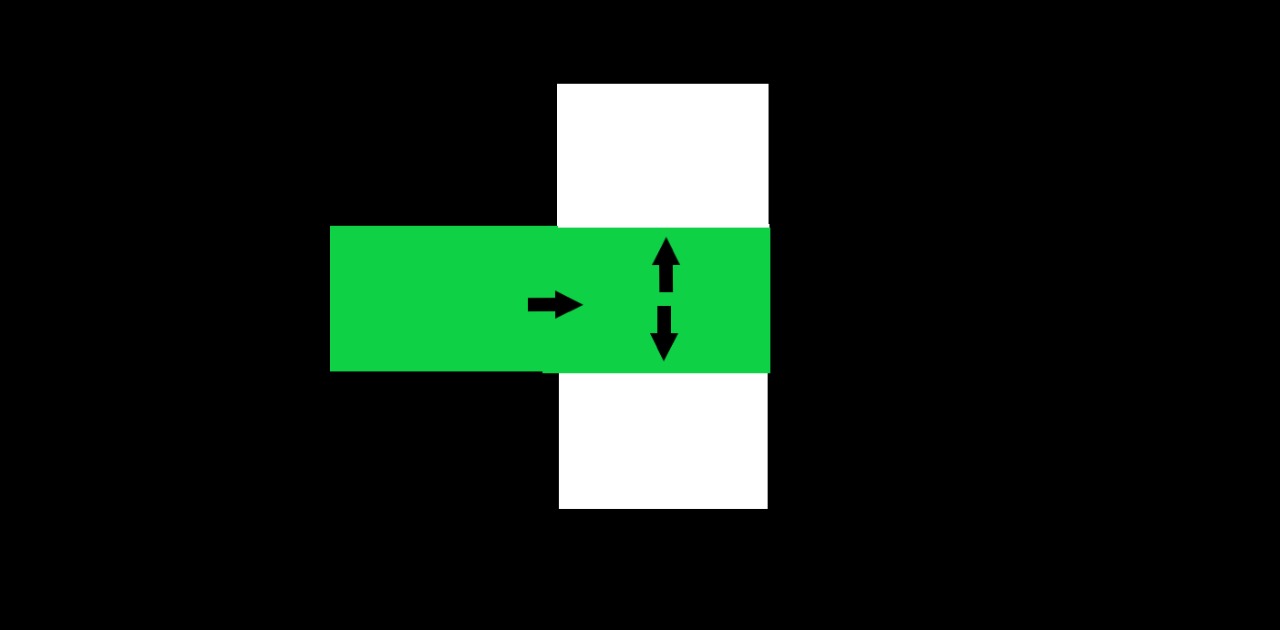
Se define la gama de colores a ocupar junto a todas la posibles direcciones que se puede tener un punto fijo a moverse.

También se le agregan las variables globales que nos ayudaran a desarrollar el ejercicio.

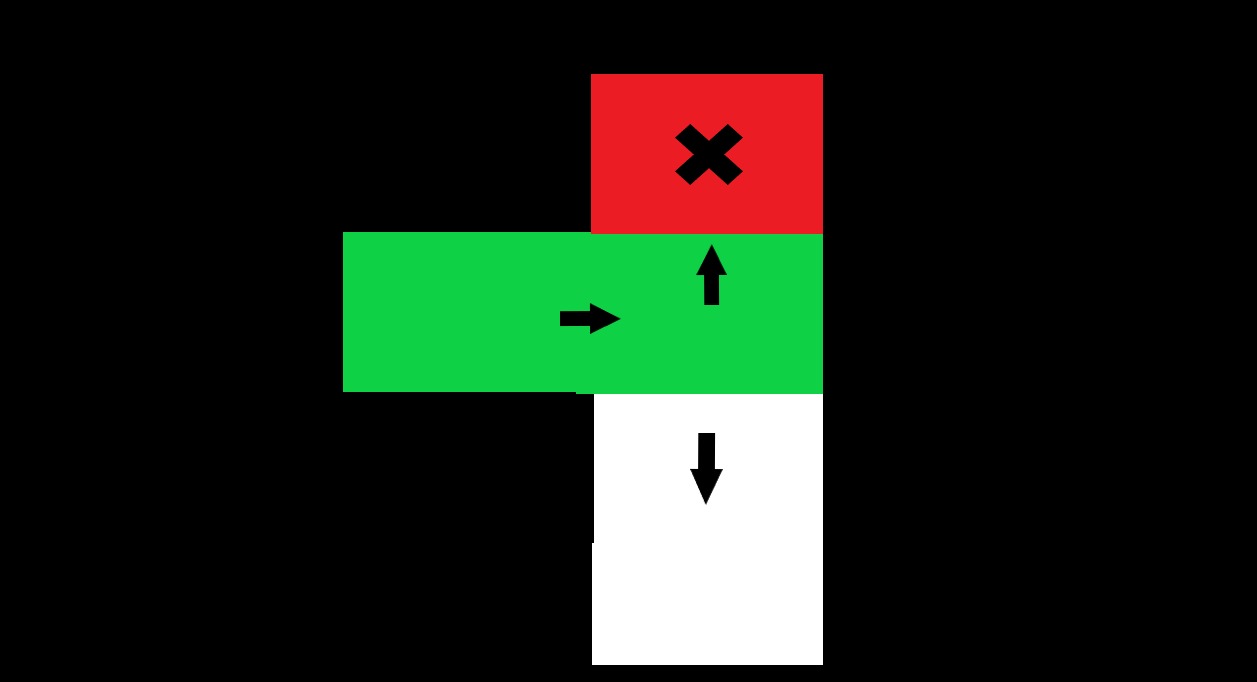
## *Solución Hilos usando semáforos y Comparativa recorrido*

En esta función se va evaluando los posibles recorridos que puede tener cada hilo y se van marcando las posiciones recorridas para no caer en loops y desperdiciar recursos.



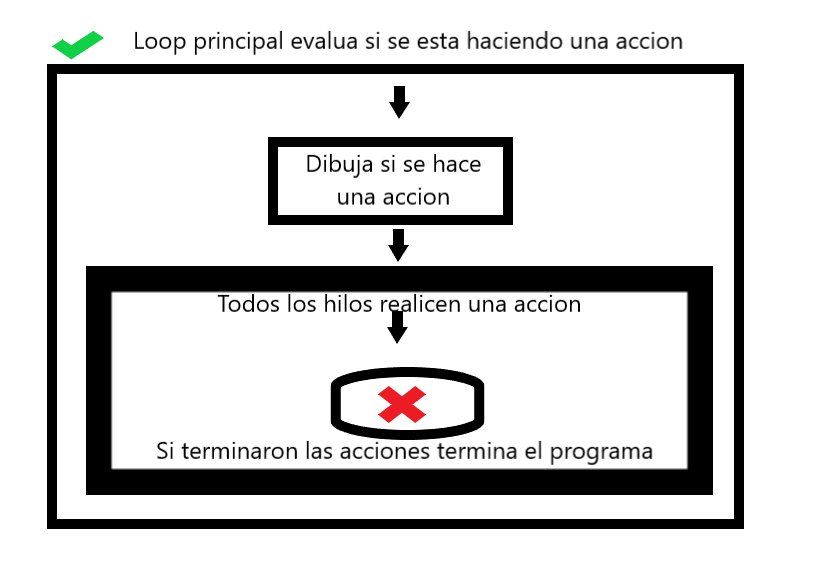


## 



## *Dibujo programa*

Al iniciar se ejecuta el siguiente ciclo, con el objetivo de dibujar el laberinto para ello ocuparemos semáforos de tipo booleanos para evaluar y múltiples hilos en uso para realizar el recorrido.



# Código documentado

**import** pygame

**from** time **import** sleep

**from** threading **import** Semaphore**,** Thread

# Gamma de colores a ocupar (para poder distingir diferentes partes del laberinto)

colores **=** **{**

' ' **:** **(**255**,**255**,**255**),**

'X' **:** **(**0**,**0**,**0**),**

'V' **:** **(**0**,**50**,**255**),**

'R' **:** **(**0**,**255**,**0**),**

'C' **:** **(**255**,**0**,**0**),**

**}**

# Todas las posibles combinaciones de espacios permitidos desde un punto fijo

direcciones**={**

'arri'**:(**0**,**1**),**

'abaj'**:(**0**,-**1**),**

'izq'**:(-**1**,**0**),**

'der'**:(**1**,**0**),**

'cent'**:(**0**,**0**)**

**}**

# Valores Globales

filas **=** 50

columnas **=** 30

run **=** **True**

encontrado**=False**

sem\_clones**=** Semaphore**(**10**)**

sem\_matriz**=** Semaphore**(**1**)**

# Valores globales de los cuadros dentro de la matriz

ancho **=** 300**/**columnas

largo **=** 300**/**filas

# Se inicia pygame

pygame**.**init**()**

# Se establecen caracteristicas basicas de la ventana (tamaño , color de fondo , contorno)

screen **=** pygame**.**display**.**set\_mode**((**600**,** 500**))**

screen**.**fill**([**255**,**255**,**255**])**

pygame**.**draw**.**rect**(**screen**,** **(**230**,**30**,**30**),** pygame**.**Rect**(**150**,**100**,**300**,**300**),** 1**)**

# Abrimos y escaneamos el archivo

file**=open(**'inputLaberinto.txt'**)**

lineas**=**file**.**readlines**()**

# Creamos la matriz

matriz **=** **[]**

**for** i **in** **range(**filas**):**

matriz**.**append**([**' '**]** **\*** 30**)**

# Colocamos las paredes y la ventana

**for** linea **in** lineas**:**

datos **=** linea**.**split**(**','**)**

columna **=** **int(**datos**[**0**])**

fila **=** **int(**datos**[**1**])**

dato **=** datos**[**2**]**

matriz**[**fila**][**columna**]** **=** dato**[**0**]**

# Funcion encargada de actualizar la pantalla del laberinto

**def** actualiza\_laberinto**():**

**for** i **in** **range(**0**,** 50**):**

**for** j **in** **range(**0**,** 30**):**

dibuja\_cuadrado**(**i**,** j**,** colores**[**matriz**[**i**][**j**]])**

pygame**.**display**.**update**()**

# Funcion encargada de ir rellenando de color las diferentes partes del laberinto

**def** dibuja\_cuadrado**(**y**,** x**,** color**):**

pygame**.**draw**.**rect**(**screen**,** color**,** pygame**.**Rect**(**150 **+** x**\***ancho **+** 1**,** 100 **+** y**\***largo **+**1**,** ancho**,** largo**))**

# Funcion encargada de contar direcciones validas donde se puede desplazar una copia

**def** cuentaBifurcaciones**(**x**,** y**):**

direccionesValidas**=[]**

**for** i **in** **[**'arri'**,** 'abaj'**,** 'izq'**,** 'der'**]:**

x\_**=**x**+**direcciones**[**i**][**0**]**

y\_**=**y**+**direcciones**[**i**][**1**]**

**if** y\_**>-**1 **and** y\_**<**columnas **and** x\_**>-**1 **and** x\_**<**filas**:**

**if** matriz**[**x\_**][**y\_**]==**' ' **or** matriz**[**x\_**][**y\_**]==**'V'**:**

direccionesValidas**.**append**(**i**)**

**return** direccionesValidas

# Funcion encargada del comportamiento de los clones dentro del laberinto

**def** clon**(**x**,** y**,** direccion**):**

**global** run

**global** encontrado

sem\_clones**.**acquire**()**

**while** run **and** **not** encontrado**:**

sleep**(**0.05**)**

x**=**x**+**direcciones**[**direccion**][**0**]**

y**=**y**+**direcciones**[**direccion**][**1**]**

sem\_matriz**.**acquire**()**

**if** matriz**[**x**][**y**]==**'V'**:**

**print(**f'La salida está en ({x}, {y})'**)**

encontrado**=True**

**return**

matriz**[**x**][**y**]=**'C'

sem\_matriz**.**release**()**

dirs**=**cuentaBifurcaciones**(**x**,** y**)**

**if** **len(**dirs**)==**1**:**

sem\_matriz**.**acquire**()**

matriz**[**x**][**y**]=**'R'

direccion**=**dirs**[**0**]**

sem\_matriz**.**release**()**

**if** **len(**dirs**)>**1**:**

sem\_matriz**.**acquire**()**

matriz**[**x**][**y**]=**'R'

sem\_matriz**.**release**()**

hilos**=[]**

**for** i **in** **range(**1**,** **len(**dirs**)):**

t**=** Thread**(**target**=**clon**,** args**=(**x**,**y**,**dirs**[**i**]))**

t**.**setDaemon**(True)**

t**.**start**()**

hilos**.**append**(**t**)**

direccion**=**dirs**[**0**]**

**if** **len(**dirs**)==**0**:**

sem\_clones**.**release**()**

**return**

t**=**Thread**(**target**=**clon**,** args**=(**0**,**0**,**'cent'**))**

t**.**setDaemon**(True)**

t**.**start**()**

# Loop principal de pygame. Es necesario para que windows no crea que no responde.

**while** run**:**

actualiza\_laberinto**()**

**for** evento **in** pygame**.**event**.**get**():**

**if** evento**.type** **==** pygame**.**QUIT**:**

run **=** **False**

pygame**.quit()**

# 

# 

# Resultados

Como se puede apreciar en los siguientes resultados este programa obtiene diversos resultados esto debido a la naturaleza de los semáforos ya que al iniciar el programa este difiere en el orden del recorrido que los hilos van a realizar.

* Salida 1:



* Salida 2:



# Conclusión

En este documento se vio el paso a paso de cómo se creó un para recorrer un laberinto ocupando un archivo txt con los datos de diseño. Se presentaron dificultades tales como: utilizar una herramienta nueva, no vista anteriormente, a su vez, crear por primera vez un programa de tales características, logrando así el objetivo.

El ejercicio sirvió para dimensionar la cantidad de información que existe, el manejo de las herramientas de lógica tales como semáforos e hilos, la capacidad de entendimiento y el desarrollo como habilidades fundamentales.

Como conclusión principal podemos encontrar que el desarrollo de un programa de tales características presenta una serie de dificultades especiales como el uso de funciones específicas del lenguaje Python para controlar el sistema.

También y por otro lado al obtener diferentes resultados se tiene que emplear una resolución el cual no vea todos los casos posibles del sistema enfatizando un ahorro de recursos.