

Curso

Principios de Programación I

Proyecto II

Laberinto

Integrantes

Gomez, Johel

Pineda, Esteban

Román, Danny

Fecha de Entrega

02 de setiembre, 2025

# Tabla de Contenidos

[Tabla de Contenidos 2](#_Toc207728899)

[Descripción del Problema 3](#_Toc207728900)

[Librerías Utilizadas 4](#_Toc207728901)

**ESTRUCTURA DEL CODIGO - CLASES Y FUNCIONES……………………………...…………………5**

[Análisis de Resultados 6](#_Toc207728902)

[Pruebas de funcionalidad 7](#_Toc207728903)

[Manual de Usuario – Laberinto 10](#_Toc207728904)

[1) Requisitos previos 10](#_Toc207728905)

[Formato de los archivos 11](#_Toc207728906)

[3) Navegación del Menú 11](#_Toc207728907)

[Repositorio de Control de versiones 14](#_Toc207728908)

[Conclusiones 15](#_Toc207728909)

# Descripción del Problema

El juego de laberintos es un juego que data de mucho tiempo atrás, inclusive en periódicos o libros impresos en papel, que han servido a lo largo del tiempo para entretener y fortalecer la mente con este juego de habilidad.

Este proyecto tal y como se indica, busca que, desde la programación, se desarrolle el juego de laberintos junto con sus reglas de juego, desarrollado en el lenguaje de Python.

Aunque no se debe de conocer tanto en cuanto a estrategias, si se debe de conocer a profundidad el juego de laberinto para poder desarrollarlo.

Se desarrollará una ventana que muestra dos opciones principales:

* **Inicia juego**: Donde el jugador puede personalizar la partida, ya sea tipo de juego normal sin importar el tiempo de duración en resolver el laberinto o bien, contra tiempo, donde dispondrá de 5 minutos para resolver el juego o bien perderá la partida. Además, para ambas opciones, se podrá elegir entre varios niveles de dificultad: Fácil, medio y difícil.
* **Ranking**: Muestra de mayor a menor, la información de los 10 juegos con mejores tiempos y los nombres respectivos de los jugadores que completan la partida. Incluyendo tiempo invertido, nombre, cantidad de movimientos y las dimensiones del laberinto seleccionado. Esta información será guardada en un archivo rankings.txt para poder cargar las mejores partidas cada vez que se ejecute el programa.

# Librerías Utilizadas

En este proyecto se utilizaron varias librerías:

1. ***tkinter***

* Esta es la librería GUI (Interfaz gráfica para el usuario por sus sigles en inglés) estándar de Python para implementar la interfaz grafica con la que el usuario jugará el laberinto.

1. ***Datetime***

* Provee las clases para manipular fechas y horas las cuales son usadas en la construcción del juego.

1. **Time**

* Usada en las funciones relacionadas con el tiempo, como *sleep* (dormir), normalmente donde se usan pausas para esperar alguna entrada del usuario para poder continuar jugando.

1. ***Threading***

* Usado para habilitar los ‘hilos’ o threads cuando se ejecutan tareas como un timer o stopwatch si detener o congelar el GUI.

# Estructuras del código – Clases y Funciones:

LABERINTO Clases: Partida: Contiene todas las variables y funciones para manejar la partida

* **Variables**
  + **partida\_iniciada**: variable boolean para establecer el estado de la partida; ayuda a controlar los botones de la interfaz para funcionar únicamente cuando la partida este activa (reiniciar partida, abandonar partida, auto completar) o inactiva (guardar\_configuraciones, iniciar partida).
  + **movimientos\_partida**: variable int para contabilizar los movimientos de cada partida; se incrementa con cada movimiento dentro del area valida del laberinto (valores 0; pared).
  + **ranking**: variable de tipo lista para almacenar el ranking cargado al iniciar la ventana.
  + **laberinto**: variable de tipo lista para almacenar el laberinto cargado de los archivos en el directorio de proyecto.
  + **bloques**: variable de tipo lista para almacenar los bloques generados al iterar el laberinto en memoria y generar los bloques con su respectivo color y valor para visualizar el laberinto.
  + **posicion\_totem**: variable que almacena la tupla con la posición del totem; se inicializa en None o nulo; al visitar la función visualizar\_laberinto(), se identifica la posición de inicio y se guarda en esta variable; el valor se cambia constantemente con base a los movimientos al estar la partida activa.
  + **modo\_seleccionado**: variable de tipo tk.StringVar() para almacenar el modo de partida seleccionado por el usuario; esta variable es de tipo tk.StringVar() para poder actualizar las etiquetas del area de juego 1 donde se muestran los detalles de la partida dinámicamente.
  + **dificultad\_seleccionada**: variable de tipo tk.StringVar() para almacenar la dificultad de partida seleccionada por el usuario.
  + **dimensiones\_seleccionadas**: variable de tipo tk.StringVar() para almacenar las dimensiones de partida seleccionadas por el usuario.
  + **contador\_tiempo**: variable de tipo tk.IntVar() para almacenar el tiempo de partida; se inicializa en 0 si la partida es de tipo Normal y el equivalente en segundos de los minutos seleccionados en caso de ser una partida de tipo Contra Tiempo.
  + **cronometro**: variable para almacenar el objeto cronometro creado al iniciar la partida.
* **Funciones**
  + **cargar\_laberinto**(): esta función se encarga de cargar el laberinto del .txt en memoria; la función toma como parámetro un String con la ruta donde se encuentra el .txt que se debe cargar en base a las configuraciones; abre el archivo en modo lectura y lo analiza linea por linea; divide las lineas entre las comas y almacena los valores en una lista que se agregara a la matriz creada para el laberinto y sus valores.
  + **cargar\_rankings**(): esta función se encarga de cargar el ranking del .txt en memoria; la función toma como parámetro un String con la ruta donde se encuentra el .txt con los rankings; abre el archivo en modo lectura y lo analiza linea por linea; divide las lineas entre las comas y almacena los valores en una lista que se agregara a la matriz creada para el laberinto y sus valores; luego de cargar el archivo en memoria, itera la lista nuevamente para encontrar los valores con mejores tiempos, mayor dificultad y menor cantidad de tiempo para ordenar el ranking en base a esto.
  + **llave\_orden**(): esta función se encarga de tomar una linea de los rankings en formato lista para retornar una tupla de 3 dígitos con el tiempo, las dimensiones y la cantidad de movimiento de la partida; en base a esta tupla se puede ordenar el ranking comparando los elementos de la lista y guardando el de menor tiempo, mayor dificultad y menos movimientos.
  + **guardar\_ranking**(): esta función se encarga de tomar la lista de rankings en memoria y guardarla en el archivo .txt para almacenar los rankings.
  + **mostrar\_ranking**(): esta función se encarga de limpiar el treeview que contiene el ranking para actualizarlo; luego de iterar los items en el treeview y eliminarlos, itera la lista de rankings y agrega cada uno de los elementos al treeview.
  + **guardar\_configuraciones**(): esta función se encarga de obtener los valores de todos los combobox del menu y actualizarlos en las variables de juego de la partida; luego de obtener los valores de los combobox “combo\_partida”, “combo\_dificultad”, “combo\_dimensiones” y “combo\_tiempo”(el valor de combo tiempo solo se valida en caso de que el tipo de partida seleccionado sea “Contra Tiempo”) procede a imprimir en la terminal las configuraciones guardadas y cargar el laberinto en base a las mismas; finalmente llama la función visualizar\_laberinto() para mostrar el laberinto en la ventana y generar los bloques del laberinto.
  + **dimensiones\_dinámicas**(): esta función se encarga de cambiar los valores del combobox de dimensiones cuando se registra un evento en el combobox de dificultad; la función toma como parámetro un evento por cada cambio en “combo\_dificultad”.
  + **visualizar\_laberinto**(): esta función se encarga de generar la matriz de bloques y demostrarlos en pantalla; comienza por eliminar todos los bloques contenidos en el cuadro del laberinto y limpia la matriz de bloques de la partida; luego genera el tamaño de las celdas en base al alto y ancho de la ventana sobre las dimensiones del laberinto; luego itera el laberinto cargado en la partida y valida si el valor (x, y) en el laberinto corresponde a -1, en ese caso guarda la posición del totem en la partida; luego genera (x1, y1) como las coordenadas de inicio y (x2, y2) como las coordenadas del final del bloque en la ventana; con estos valores la función crea el rectángulo para representar el bloque y lo agrega a la fila de bloques que se agregara como sublista a la matriz de bloques.
  + **mostrar\_totem**(): esta función se encarga de visualizar el totem en la pantalla según las posiciones donde se encuentra el inicio del laberinto o en base a los movimientos del usuario sobre el laberinto; la función toma como parámetro la posición (x, y) sobre la que se debe mostrar el totem; esta función borra el totem del cuadro donde están contenidos los bloques del laberinto y lo vuelve a mostrar con las nuevas coordenadas recibidas como parámetro.
  + **movimiento**(): esta función se encarga de mover el totem dentro del laberinto; la función toma como parámetro un evento generado por el input de presionar una tecla del teclado cuando la partida este iniciada; al iniciar, la función borra el totem del cuadro de laberinto y obtiene la posición actual de la partida; en base al tipo de evento (Up, Down, Left, Right) se cambiaran las coordenadas del totem y se muestra nuevamente en pantalla; la función también valida el caso de haber llegado al final cuando se encuentra con el valor 2 en el laberinto; en ese punto genera una ventana nueva para obtener el nombre del jugador para guardar la partida.
  + **resolver\_laberinto**(): esta función se encarga de recorrer los caminos del laberinto y determinar si cada uno de los bloques de camino lleva a la solución del laberinto; la función toma como parámetros el laberinto cargado en la partida, la posición de fila y columna del totem y una matriz nueva generada únicamente para iterar los caminos en la validación; la función logra encontrar el camino correcto validando que la posición del totem se mantenga dentro de los limites del laberinto (paredes y final de la matriz) y si la posición encontrada es la posición final; por cada uno de los bloques, se comienza una búsqueda por cada uno de los caminos disponibles; esta búsqueda se logra con una iteración recursiva que documenta todos los bloques hasta llegar al final del laberinto; en caso de no llegar al final del laberinto, retorna false y continua en el siguiente bloque con potencial de llegar al final de la partida; la función retorna False en caso de iterar todo el camino y marcar todo el laberinto como visitado sin encontrar la salida.
  + **iniciar\_partida**(): esta función se encarga de iniciar la partida; la función solo opera cuando la partida esta iniciada; para iniciar la partida los movimientos de la partida se establecen en 0, se cambia el estado de partida\_iniciada a True, se muestra el laberinto a jugar en la ventana y se muestra el totem en la posición de inicio; en caso de ser una partida Normal, el cronometro se crea de tipo progresivo, y en caso de ser una partida Contra Tiempo, se crea un cronometro de tipo regresivo.
  + **reiniciar\_partida**(): esta función se encarga de retornar todo el laberinto y juego a su estado inicial para volver a comenzar la partida; esto se logra cambiando el estado de la partida a False, deteniendo el cronometro activo, recargando las configuraciones de partida y llamando a la función iniciar\_partida nuevamente.
  + **autocompletar**(): esta función se encarga de auto completar el laberinto en pantalla en caso de existir una solución valida para el laberinto en base a la salida de la función resolver\_laberinto(); para esto cambia el estado de la partida a False y detiene el cronometro ya que la partida no esta siendo controlada por un usuario en ese momento; luego crea una matriz nueva en base al resultado de la función resolver\_laberinto() y en caso de esta matriz tener un resultado valido, se itera la matriz y por cada bloque visitado en la resolución del laberinto, se mueve el totem en base a sus coordenadas y se cambia el color del bloque en pantalla.
  + **abandonar\_partida**(): esta función se encarga de abandonar la partida en caso de que el usuario así lo decida; esto se logra cambiando el estado de partida\_iniciada a False, luego se detiene el cronometro y se vuelve a cargar las configuraciones de la partida en la ventana; para limpiar el progreso de la partida se vuelve a visualizar el laberinto limpio en pantalla.
  + **finalizar\_partida**(): esta función se encarga de finalizar la partida en caso de que el laberinto sea completado o que el tiempo de partida se acabe en las partidas contra tiempo; esto se logra cambiando el estado de partida\_iniciada a False, luego en caso de que la partida sea ganada y únicamente cuando el usuario haya ingresado un nombre para guardar la partida, se guarda la partida en los rankings y se guardan en el .txt; luego de guardarlos se vuelven a cargar los rankings en la partida y se actualiza el treeview de rankings; finalmente se vuelven a cargar las configuraciones de partida y se limpia el laberinto. La ventana emergente para indicar al usuario que la partida ha sido ganada o perdida se cierra al final de la función.

Bloque: Contiene las variables y funciones para almacenar los bloques del laberinto; al generar el laberinto se guardan todos los bloques creados en una variable de la partida.

* **Variables**
  + **cuadro:** variable para almacenar el cuadro de tkinter donde se va a visualizar la matriz de bloques.
  + **bloque**: variable para almacenar el cuadro de tkinter que va a representar el bloque.
  + **fila**: variable de tipo int para almacenar el valor que representa la posición en las filas de la matriz.
  + **columna**: variable de tipo int para almacenar el valor que representa la posición en las columnas de la matriz.
  + **valor**: variable para almacenar el valor asignado al bloque en base al laberinto (-1, 0, 1, 2)
  + **visitado**: variable de tipo boolean para establecer cuando el bloque ya ha sido visitado y cambiar su color en base
* **Funciones**
  + **set\_color**(): esta función se encarga de cambiar el color del bloque utilizando la función itemconfigure() del cuadro del bloque; esta función recibe como parámetro un string con el color a configurar.
  + **get\_color**(): esta función se encarga de retornar el color que esta almacenado en el bloque, de esta manera se puede conocer el color de cada bloque que se tiene que generar en la ventana para el laberinto.

**Cronometro**: Contiene las variables y funciones para manejar el cronometro de juego; al iniciar una partida siempre se crea un objeto de tipo cronometro para iniciar y detener el cronometro.

El cronometro se logra creando un proceso con la función runtime de la misma clase.

* **Variables**
  + **partida**: variable para almacenar el objeto partida sobre el cual se esta creando el cronometro; esto permite acceso a la variable “contador\_tiempo” de la partida para actualizarla con el paso del tiempo en el proceso.
  + **segundos**: esta variable de tipo int almacena la cantidad de segundos que tiene configurada la partida sobre la cual se crea el cronometro; se incrementa o disminuye según el paso del tiempo en base al modo de juego; “Normal” se incrementa y “Contra Tiempo” se disminuye la cantidad de segundos.
  + **tipo\_crono**: esta variable de tipo String almacena el tipo de cronometro creado; puede ser de tipo “progresivo” para partidas de tipo “Normal”, o “regresivo”para partidas de tipo “Contra Tiempo”.
  + **corriendo**: variable de tipo Boolean para establecer si el cronometro esta activo o no; al estar activo la función runtime() se ejecuta.
  + **proceso**: variable para almacenar el proceso y poder inicializarlo
* **Funciones**
  + **iniciar**(): esta función se encarga de cambiar la variable “corriendo” a True, luego crea el proceso sobre el que va a contabilizar los segundos y lo comienza con la función .start() de la librería threading; en base a la variable “tipo\_crono” se incrementan o disminuyen los segundos de la partida.
  + **runtime**(): esta función se encarga de proporcionar el código que se va a cargar en el hilo creado para llevar el tiempo en el cronometro; la función tiene un ciclo que solo se ejecuta cuando cuando la variable “corriendo” esta configurada en True; dentro de este ciclo una condicional if-elif suma o resta 1 a los segundos dependiendo del valor de la variable “tipo\_crono” y finalmente se encarga de terminar la partida por tiempo en caso de ser un cronometro regresivo; al llegar a 0 aparece una ventana indicando al usuario que la partida termino y se acabo el tiempo.
  + **detener**(): esta función se encarga de cambiar la variable “corriendo”a False lo que resulta en la terminación del proceso del cronometro

# Análisis de Resultados

**Objetivos Alcanzados:**

Al ejecutar el programa de Laberinto, se realizaron diferentes pruebas para comprobar y garantizar su correcto funcionamiento.

En lo que a modo de juego se refiere, se realizaron pruebas con los niveles fácil, medio y difícil, tanto, contra tiempo cómo en modo normal sin contra reloj.

En cuanto a la opción de ranking, se verificó que el mismo si guarda correctamente los 10 mejores tiempos de juego en la resolución del laberinto, de forma descendente.

También contra-reloj se realizaron las pruebas correspondientes y se logró observar que cumple a cabalidad los 5 minutos de expiración, donde si el jugador no alcanzó la meta del laberinto dentro del tiempo establecido, pierde el juego.

**Objetivos No alcanzados:**

Dentro de los objetivos no alcanzados podemos mencionar que no logramos realizar a tiempo, una función que nos permitiera autocompletar los laberintos sin importar el tipo de laberinto ingresado.

# Pruebas de funcionalidad

* **Fácil:**  
    
  A screenshot of a game

  AI-generated content may be incorrect.  
    
  A screenshot of a computer

  AI-generated content may be incorrect.
* **Medio:**

A screenshot of a game

AI-generated content may be incorrect.

* **Difícil:**

A screenshot of a game

AI-generated content may be incorrect.

# Manual de Usuario – Laberinto

## 1) Requisitos previos

* Python 3 instalado.
* IDLE (viene con Python).
* En la misma carpeta del archivo P2\_Johel\_Esteban\_Danny.py debe existir un archivo de texto y una carpeta que contiene a su vez tres subcarpetas o subfolders, uno para contener los laberintos txt de nivel fácil, otra de nivel intermedio y uno más para nivel difícil:
* ranking.txt
* Folders:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

### Formato de los archivos

* *rankings.txt* → un jugador por fila, con formato Nombre del jugador, tiempo invertido, dimensiones y cantidad de movimientos.

***2) Cómo ejecutar el programa en IDLE***

1. Abrir IDLE
2. Menú > File > Open > Seleccionar archivo ***P2\_Johel\_Esteban\_Danny.py***
3. Con el archivo abierto > Run > Run Module

## 3) Navegación del Menú

Al ejecutar programa, se mostrará el siguiente menú principal:

*\*\*\* Menú \*\*\**

*Elija una de las siguientes opciones:*

*1. Modo de juego: normal ó contra tiempo:  
  
A blue and white stripe

AI-generated content may be incorrect.*

*2. Dificultad: Fácil, medio ó difícil:*

*A blue and white stripes

AI-generated content may be incorrect.*

*3. Dimensiones: 8x8, 9x9, 10x10, 11x11,12x12, 13x13, 14x14, 15x15 ó 16x16:*

*A blue and white rectangle with black text

AI-generated content may be incorrect.*

*Una vez seleccionados una de cada de las diferentes opciones, se procederá a seleccionar “guardar configuración” para guardar la configuración seleccionada:*

*A close up of a sign

AI-generated content may be incorrect.*

*Posteriormente se debe de seleccionar la opción de ‘iniciar’ para comenzar el juego:*

*A screenshot of a phone

AI-generated content may be incorrect.*

*Además, como adicional se cuenta con las funcionalidad de reiniciar la resolución del laberinto:*

*A screenshot of a phone

AI-generated content may be incorrect.*

*La opción de Auto completar, para poder visualizar la ruta de resolución del laberinto en ejecución:*

*A screenshot of a phone

AI-generated content may be incorrect.*

*y por último, la opción de Abandonar, para retirarse del juego de laberinto en ejecución:*

*A screenshot of a phone

AI-generated content may be incorrect.*

# 

# Repositorio de Control de versiones

Para este proyecto, se creó un repositorio en GitHub, el cuál puede accederse desde el siguiente enlace: <https://github.com/pinesandev/Proyecto2-Principios1>

# Conclusiones

El desarrollo del programa *Laberinto* permitió que aplicaramos de forma práctica los conceptos fundamentales de programación en Python vistos en este curso Principios de programación I, tales como el uso de listas, funciones, estructuras condicionales, bucles y la manipulación de archivos de texto para la carga de laberintos y rankings.

Hemos logrado cumplir con los objetivos propuestos:

* Implementar una interfaz gráfica funcional mediante la librería *tkinter,* que permite al usuario configurar y jugar las partidas de forma intuitiva.
* Ofrecer dos modos de juegos: Normal y Contra tiempo, con diferentes niveles de dificultad y dimensiones personalizables.
* Registrar y modificar un raking de los mejores tiempos, movimientos y configuraciones de los jugadores, almacenado en un archivo .txt.
* Representar el funcionamiento de un cronómetro progresivo o contra reloj, según el tipo de partida elegida, utilizando hilos (threads) para no afectar la interfaz y experiencia del usuario.

Este proyecto además sirvió como ejercicio para aplicar el razonamiento básico así como implementar buenas prácticas de programación, como lo son la modularización del código, el uso controlado de variables globales, validación de entradas del usuario y presentación de mensajes y resultados en pantalla.

Además, fue necesario realizar investigaciones adicionales complementarias para resolver dudas, como por ejemplo con el profesor para asegurarnos de cumplir los parámetros y requisitos necesarios. Además de investigación de *widgets* de *tkinter* y manejo de hilos (threading).

Para concluir, podemos decir que se ha generado un programa funcional y estable, para que este pueda ser utilizado por cualquier usuario y disfrutar de una experiencia de juego entretenido, al mismo que se puso en práctica habilidades de razonamiento y programación.