Así que acabas de empezar a aprender Python y quieres desarrollar un proyecto interesante después de aprender lo básico. Pues bien, esta serie de videos es exactamente para ti, porque en esta serie vamos a desarrollar un juego con Python.

Ahora, si ya sabes lo básico y tomaste algún curso introductorio —no importa de qué canal o curso sea— entonces esta es la serie que estabas buscando, porque te dará las bases de cómo comenzar a estructurar un proyecto en Python.

En esta serie desarrollaremos un juego llamado minesweeper, que es conocido como un juego de un solo jugador, muy divertido y también desafiante. Veamos qué juego aprenderás a desarrollar en esta serie.

Bien, ese será el juego que desarrollaremos a lo largo de esta serie. Como puedes ver, tenemos varias celdas en las que podemos hacer clic para abrirlas. El objetivo principal en Minesweeper es no hacer clic en una celda que tenga una mina detrás.

Eso significa que, si abro una celda, puedo recibir un número. Ese número indica que en las celdas alrededor de esa celda hay una mina cerca que no debo tocar. Por ejemplo, puedo marcar una mina sospechosa con el clic derecho del ratón. Si hago clic izquierdo en una mina, pierdo el juego.

El reto es adivinar todas las celdas seguras y evitar las que tienen minas. Si logras hacerlo, ganas el juego. Esto requiere mucha estrategia y pensamiento, lo que hace que el juego sea divertido y desafiante.

Antes de comenzar, sería genial que hicieras clic en el botón de "me gusta" y compartieras este video, especialmente si conoces a personas que recién empiezan a aprender Python. Esta serie también podría ser muy útil para ellas.

Comenzando con la Interfaz Gráfica del Juego (GUI)

Antes de escribir el proyecto, asegúrate de tener Python instalado y un IDE listo. Yo estoy usando Python 3.8, pero cualquier versión posterior debería funcionar. No usaremos muchas librerías externas, solo Tkinter, que es excelente para crear aplicaciones gráficas en Python.

Empezamos importando todo desde Tkinter. Luego creamos una ventana básica con root = Tk(). Para hacer que la ventana se ejecute, llamamos a root.mainloop(). Podemos cambiar propiedades de la ventana como tamaño (geometry()), título (title()) y color de fondo (configure()).

También deshabilitamos el cambio de tamaño con root.resizable(False, False). Esto facilita manejar el diseño cuando agregamos elementos.

Creando Frames para Dividir la Ventana

Usamos Frames para organizar la ventana en secciones. Por ejemplo:

Un frame superior para el título.

Un frame izquierdo para una barra lateral (como un marcador de puntuación).

Un frame central para la cuadrícula del juego.

Los frames se colocan con el método place() usando coordenadas en píxeles (x, y). Al inicio usamos valores fijos, pero más adelante los haremos dinámicos.

Posicionando Elementos Dinámicamente

Para evitar escribir números fijos, creamos un archivo settings.py para guardar constantes como WIDTH, HEIGHT y el tamaño de la cuadrícula. También añadimos un archivo utils.py con funciones auxiliares height\_prct() y width\_prct() para calcular dimensiones como porcentajes del tamaño de la ventana.

Esto hace que el diseño sea adaptable y más fácil de ajustar. Por ejemplo, en lugar de poner fijo 180 píxeles de altura para un frame, podemos escribir utils.height\_prct(25) para usar el 25% de la altura de la ventana.

Al final de este tutorial, tenemos una ventana gráfica estructurada con tres frames (superior, izquierdo y central), todos organizados y con estilo. En el próximo episodio añadiremos elementos interactivos para comenzar a construir la lógica del juego.

Este es el segundo episodio de la serie sobre desarrollo de juegos en Python. En este, comenzaremos a crear clases y métodos, además de implementar conceptos de programación orientada a objetos.

Para que el juego Buscaminas funcione, necesitamos crear celdas dinámicamente. Esto requiere lógica para garantizar que la ventana sea lo más dinámica posible.

Creación de una clase de celda

Comenzamos introduciendo una nueva clase llamada `Cell`. Cada celda tendrá atributos como:

Si contiene una mina.

Si está abierta o cerrada.

Aunque se podría crear un botón directamente en Tkinter, creamos nuestra propia clase `Cell` para organizar mejor los atributos y el comportamiento. El objeto de botón de Tkinter se asocia entonces a cada celda.

El constructor `\_\_init\_\_` establece valores predeterminados, por ejemplo, `is\_mine=False`. Posteriormente, creamos métodos que generan objetos de botón dentro de cada celda.

Posicionamiento dinámico de botones

Inicialmente, probamos a colocar botones con el método `place()`. Sin embargo, esto se complica al crear muchas celdas, ya que debemos calcular manualmente la posición de los píxeles.

Para simplificar, cambiamos al método `grid()`. Con `grid()`, el marco principal se divide en filas y columnas comenzando desde cero, lo que facilita enormemente la alineación dinámica de los botones.

Al anidar dos bucles `for`, generamos una cuadrícula de botones (celdas). Por ejemplo, iterar por las coordenadas `x` e `y` nos permite generar automáticamente un tablero completo de Buscaminas.

También introducimos la constante `GRID\_SIZE` en el archivo `settings.py`. Este valor define cuántas filas y columnas tendrá el tablero, lo que aumenta la flexibilidad del juego.

Asignación dinámica de eventos (clics izquierdo y derecho)

En Buscaminas, los clics izquierdo y derecho tienen significados diferentes:

Clic izquierdo: abre una celda.

Clic derecho: marca una celda como una mina potencial.

Utilizamos el método `bind()` en Tkinter para gestionar estos eventos. Por ejemplo:

`<Button-1>` representa un clic izquierdo.

-`<Button-3>` representa un clic derecho.

En la clase `Cell`, creamos los métodos `left\_click\_actions` y `right\_click\_actions`, cada uno con un parámetro `event`. Esto nos permite gestionar la entrada del usuario y responder en consecuencia.

Al hacer clic en una celda, el programa imprime mensajes en la consola como "He hecho clic izquierdo" o "He hecho clic derecho", confirmando que los eventos funcionan.

Conclusión

Al final de este episodio, tendremos:

Una clase "Cell" con atributos y métodos.

Una cuadrícula dinámica de botones creada con bucles anidados.

Gestión de eventos para clics izquierdo y derecho.

En el próximo episodio, añadiremos más lógica para gestionar las minas dentro de las celdas, acercándonos así a un juego de Buscaminas completamente funcional.

En el tercer episodio trabajamos con lógica más compleja para Minesweeper.   
Se agregan más atributos y métodos para extender el juego y ver el comportamiento esperado.  
  
Este episodio y el siguiente se titulan “Algoritmos de Buscaminas” ya que escribiremos   
código para calcular la colocación de minas y la lógica alrededor de las celdas.  
  
  
  
Incrementar el tamaño de las celdas  
Se aumentó el tamaño de los botones (ancho y alto) en la clase `Cell` para que sean más   
visibles y cómodos al jugar.  
  
Atributos x e y  
Se añadieron los atributos `x` y `y` a cada celda para identificar su posición única en la   
rejilla. Esto permite saber, por ejemplo, que la celda en la esquina tiene coordenadas (0,0)   
y la última en un tablero 6x6 es (5,5).  
  
Organizar todos los objetos en un solo lugar  
Se creó un atributo de clase `all` que almacena todas las instancias de celdas. Esto facilita   
gestionar las 36 celdas del tablero.  
  
Se sobrescribió el método mágico `\_\_repr\_\_` para mostrar representaciones más amigables de los   
objetos en consola, mostrando sus coordenadas en lugar de IDs.  
  
Convertir celdas en minas aleatoriamente  
Se implementó el método estático `randomize\_mines`, que usa la librería `random` para seleccionar   
al azar un número de celdas y convertirlas en minas. El número de minas se calcula en `settings.py`   
como un cuarto del total de celdas.  
  
Ejemplo:   
 En un tablero 6x6 → 36 celdas → 9 minas.   
 En un tablero 7x7 → 49 celdas → 12 minas (con división entera).  
  
Conclusión  
El episodio concluye mostrando cómo estas implementaciones preparan la base para los próximos pasos:   
detectar minas alrededor de las celdas y mejorar la visualización al hacer clic en ellas.