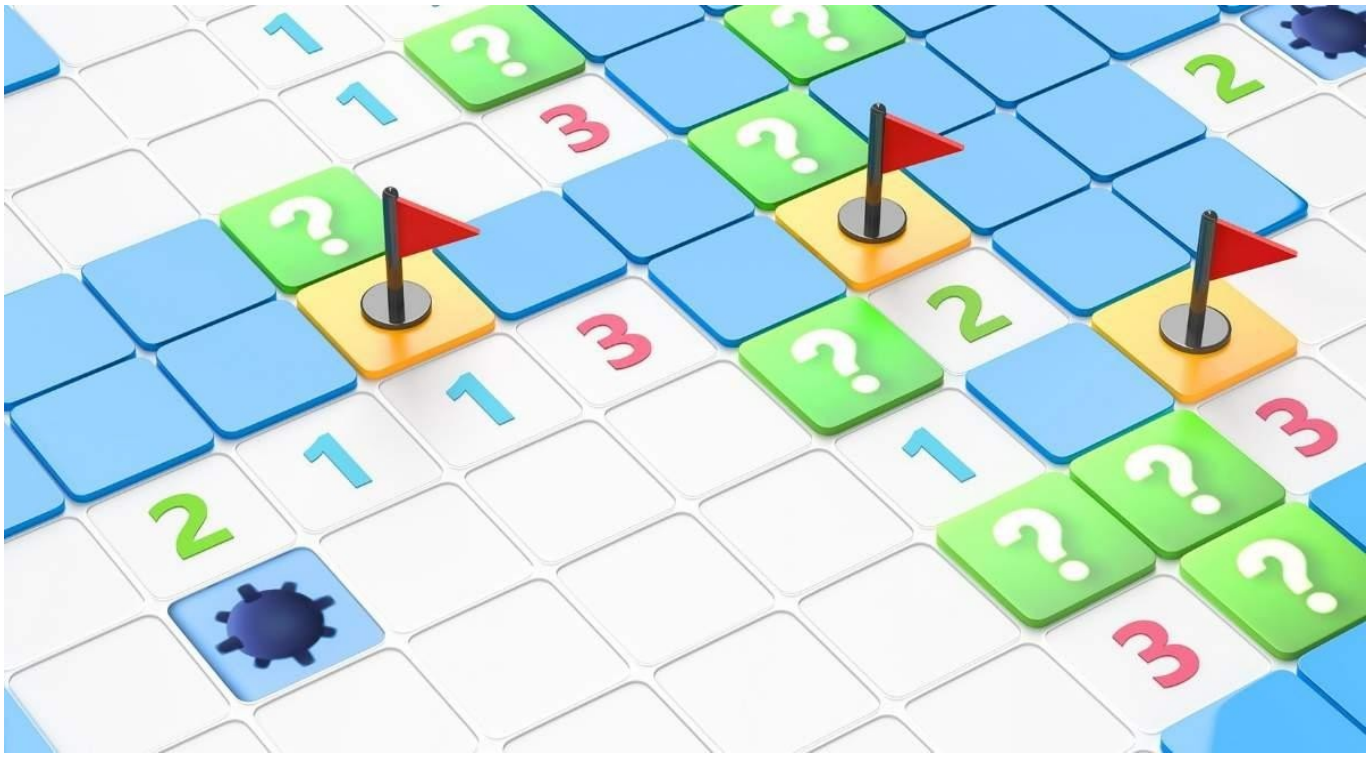


BUSCAMINAS CON TKINTER

UT - 3. 1. BUSCAMINAS CON TKINTER

Apellidos: Alves Mascareña

Nombre: María

Nº PC: Portátil

Centro: I.E.S. La Marisma (Huelva)

Curso: 2º Técnico Superior en Desarrollo de Aplicaciones Web

Asignatura: Horas de Libre Configuración – Python

Profesor: Gonzalo Cañadillas Rueda

Fecha: 09/02/2025

Sumario

Descripción del proyecto

Este proyecto consiste en la recreación del clásico juego “Buscaminas” usando la librería tkinter

El juego cuenta con tres dificultades que cambian el tamaño del tablero y el número de minas a buscar. Las partidas son puntuadas por tiempo, con un temporizador que comienza con el primer click y finaliza cuando gana o pierde, si gana, dicho tiempo se guardará en un archivo .json de records, los cuales son mostrados en un apartado “Records” en la propia interfaz del juego. Usaremos las siguientes librerías:

```python
import tkinter as tk
from tkinter import ttk, messagebox
import random
import time
import json
import os
```

- Tkinter, ttk y messagebox para la interfaz gráfica
- Random para la generación aleatoria de posiciones de minas
- Time para el temporizador
- Json y os para cargar y guardar el archivo de records

Clase: Celda

Esta clase representará cada una de las celdas en el tablero, son, en esencia, botones con propiedades extras

Propiedades

```python
# Clase que representa una celda del tablero
class Celda:
    def __init__(self, x, y, boton):
        self.x = x
        self.y = y
        self.boton = boton
        self.tiene_mina = False
        self.revelada = False
        self.marcada = False
        self.numero_adyacente = 0
```

- x: Coordenada x de la celda, se recibe por parámetro
- y: Coordenada y de la celda, se recibe por parámetro
- boton: El botón asociado a la celda, se pasa por parámetro
- tiene_mina: Booleano que indica si dicha celda tiene o no una mina, inicializado en False
- revelada: Booleano que indica si la celda ha sido revelada, inicializado en False
- marcada: Booleano que indica si la celda ha sido marcada con una bandera, inicializado en False
- numero_adyacente: Valor que representa la cantidad de minas adyacentes a esa celda, inicializado a 0

La clase celda no cuenta con métodos

Clase: Buscaminas

En esta clase construiremos nuestro juego implementando la clase anterior

Propiedades

```python
# Clase que representa el juego de Buscaminas
class Buscaminas:
    def __init__(self, filas=16, columnas=16, minas=40):
        self.filas = filas
        self.columnas = columnas
        self.minas = minas
        self.celdas = []
        self.minas_restantes = minas
        self.tiempo_inicio = None
        self.temporizador_activo = False
        self.records = {"Principiante": [], "Intermedio": [], "Experto": []}
```

● filas: Número de filas que tiene el tablero, se recibe por parámetro, si no recibe nada su valor será 16

● columnas: Número de columnas que tiene el tablero, se recibe por parámetro, si no recibe nada su valor será 16

● minas: Número de filas que hay en el tablero, se recibe por parámetro, si no recibe nada su valor será 40

Estos tres valores que le pasamos por defecto si no recibe ningún parámetro lo usaremos a nuestro favor para inicializar el juego en la dificultad “Intermedio”, ya que estos valores coinciden con dicha dificultad

● celdas: Lista de las celdas que estarán en nuestro tablero

● minas_restantes: Número de minas que quedan sin marcar en el tablero, inicializado al número de minas totales

● tiempo_inicio: Tiempo al inicio del juego, inicializado a None, ya que comenzaremos a 00:00

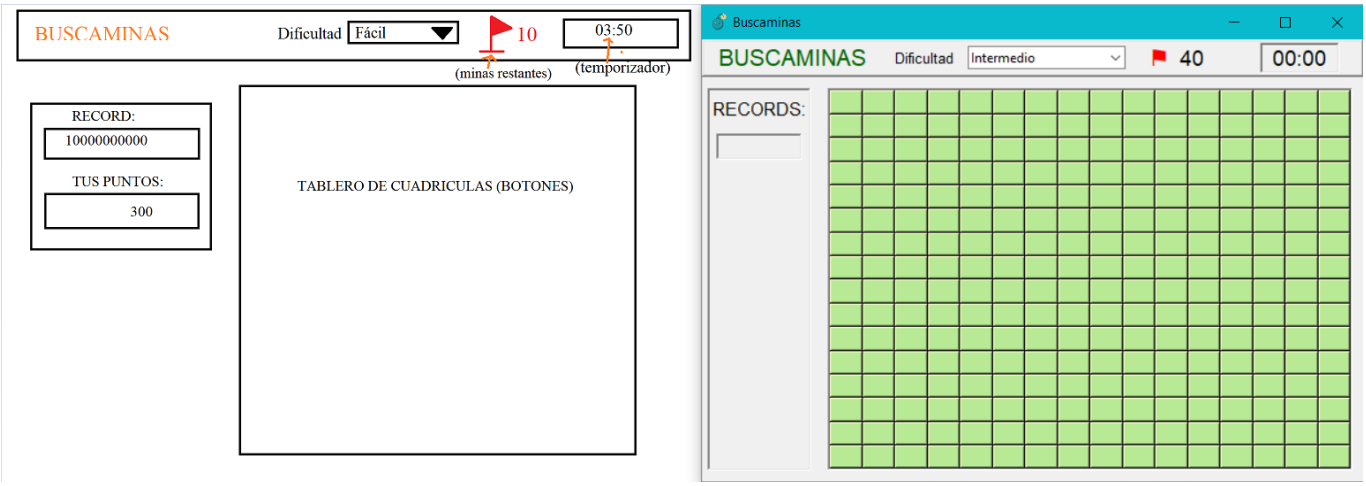
● temporizador_activo: Booleano que nos indicia que si el temporizador está avanzando, inicializado a False, ya que no comenzará hasta el primer clic

● records: Diccionario con listas para cada dificultad, donde se almacenarán los records de la sesión actual y posteriormente se guardarán en el archivo .json

Métodos

► crear_interfaz(self)

Para aprovechar las posibilidades gráficas de Tkinter de manera óptima sin limitarme a los pocos conocimientos que poseo por ser algo nuevo para mí, le pedí al asistente IA, ChatGPT, que recreara el dibujo que le proporcioné en tkinter, a lo cual, después de un par de modificaciones, quedó así



Implementamos el código recibido en el método, detallaremos los elementos por partes:

Ventana

```python
# Método para crear la interfaz gráfica
def crear_interfaz(self):
    self.ventana = tk.Tk()
    self.ventana.title("Buscaminas")
    self.ventana.geometry("600x400")
    self.ventana.iconbitmap("img/icono.ico")
    self.ventana.minsize(600, 400)
    self.ventana.maxsize(600, 400)
```

- Comenzamos creando una ventana con .Tk

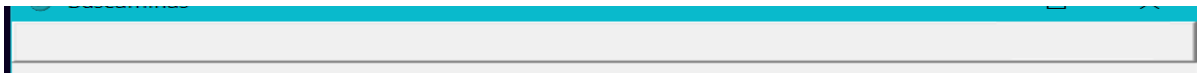
- Personalizamos el título y el icono

- Establecemos el tamaño de la ventana, los mínimos y máximos, como tienen el mismo valor conseguimos que la ventana no sea redimensionable

Encabezado

```python
# Crear el encabezado
self.encabezado = tk.Frame(self.ventana, relief=tk.RAISED, borderwidth=2)
self.encabezado.pack(side=tk.TOP, fill=tk.X)
```

Creamos un Frame dentro de ventana y lo colocamos en la parte superior, haciendo que ocupe todo lo posible horizontalmente. Le añadimos relieve y borde



Título

```
# Título del juego
self.titulo = tk.Label(self.encabezado, text="BUSCAMINAS", font=("Arial", 16), fg="darkgreen")
self.titulo.pack(side=tk.LEFT, padx=10)
```

Label dentro del encabezado situado a la derecha. Personalizamos el contenido, la fuente y el color

Selector de dificultad

```
# Selector de dificultad
self.dificultad_label = tk.Label(self.encabezado, text="Dificultad", font=("Arial", 10))
self.dificultad_label.pack(side=tk.LEFT, padx=10)

self.dificultad_selector = ttk.Combobox(self.encabezado, values=["Principiante", "Intermedio", "Experto"], state="readonly")
self.dificultad_selector.set("Intermedio")
self.dificultad_selector.pack(side=tk.LEFT)
self.dificultad_selector.bind("<<ComboboxSelected>>", lambda event: self.cambiar_dificultad())
```

Creamos otro label dentro del encabezado donde escribiremos “Dificultad” colocado a la derecha.

Seguimos con un selector Combobox con las opciones “Principiante”, “Intermedio” y “Experto”, establecemos su estado a “readonly” para que no sean modificables sus opciones, lo colocamos a la derecha y le enlazamos al cambio de selección el evento lambda con la función cambiar_dificultad()

Minas

```
# Frame para mostrar las minas restantes
self.minas_frame = tk.Frame(self.encabezado)
self.minas_frame.pack(side=tk.LEFT, padx=20)

self.bandera_icono = tk.Label(self.minas_frame, text="\u2691", font=("Arial", 16), fg="red")
self.bandera_icono.pack(side=tk.LEFT)

self.minas_restantes_label = tk.Label(self.minas_frame, text=str(self.minas_restantes), font=("Arial", 14))
self.minas_restantes_label.pack(side=tk.LEFT, padx=5)
```

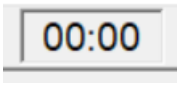
Creamos un nuevo frame dentro del encabezado donde colocaremos el label para la bandera y el label para las minas restantes

El valor del label de minas_restantes está asociado a la variable minas_restantes, por lo cual, cuando esta variable cambie, el label cambiará también

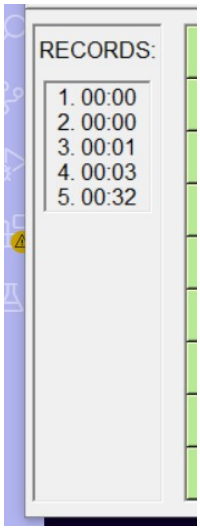
Temporizador

```
# Temporizador
self.temporizador = tk.Label(self.encabezado, text="00:00", font=("Arial", 14), relief=tk.SUNKEN, width=6)
self.temporizador.pack(side=tk.RIGHT, padx=20)
```

Creamos un label para el temporizador que inicializaremos al 00:00, le damos relieve consiguiendo el borde



Puntaje



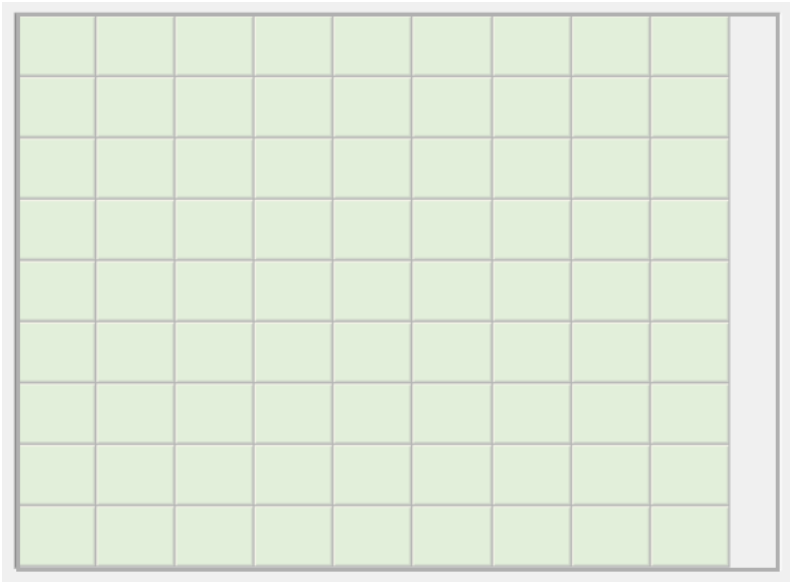

```
# Puntajes
self.puntajes_frame = tk.Frame(self.ventana, relief=tk.SUNKEN, borderwidth=2)
self.puntajes_frame.pack(side=tk.LEFT, fill=tk.Y, padx=5, pady=10)

self.record_label = tk.Label(self.puntajes_frame, text="RECORDS:", font=("Arial", 12))
self.record_label.pack(pady=5)

self.record_puntos = tk.Label(self.puntajes_frame, text="", font=("Arial", 12), relief=tk.SUNKEN, width=8)
self.record_puntos.pack(pady=5)
```

Empezamos creando un frame a la derecha que ocupará el espacio total verticalmente, aquí almacenaremos ambos labels, record_label que será el título y record_puntos donde mostraremos las puntuaciones

Frame del Tablero

```
# Frame del tablero
self.frame_tablero = tk.Frame(self.ventana, relief=tk.SUNKEN, borderwidth=2)
self.frame_tablero.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=10, pady=10)
```

El espacio donde colocaremos nuestras casillas, este frame ocupará el espacio restante expandiéndose por ambos ejes



► crear_tablero(self)

Este método prepara el tablero para jugar

```python
def crear_tablero(self):
    for widget in self.frame_tablero.winfo_children():
        widget.destroy()  # Eliminar los widgets existentes en el frame del tablero

    # Configurar el grid para que se expanda
    for i in range(self.filas):
        self.frame_tablero.grid_rowconfigure(i, weight=1)
    for j in range(self.columnas):
        self.frame_tablero.grid_columnconfigure(j, weight=1)

    # Crear los botones del tablero
    for x in range(self.filas):
        fila = []
        for y in range(self.columnas):
            boton = tk.Button(self.frame_tablero, width=3, height=1, command=lambda x=x, y=y: self.revelar_celda(x, y))
            boton.bind("<Button-3>", lambda e, x=x, y=y: self.marcar_celda(x, y))
            boton.grid(row=x, column=y, sticky="nsew")  # Botones expandidos
            boton.config(bg="#b8ea96")

            # Ajuste adicional para la dificultad Principiante
            if self.dificultad_selector.get() == "Principiante":
                boton.config(width=5, height=2)  # Ajuste del tamaño de las casillas

            fila.append(Celda(x, y, boton))
        self.celdas.append(fila)
```

Primero vacía todo lo que exista en el frame de nuestras casillas, configura el tamaño del grid según el número de filas y columnas, así conseguimos que las casillas ocupen el mayor espacio posible independientemente del número que sean

Seguimos con la creación de las casillas, creamos un bucle for anidado, inicializando la lista “fila” en el primer bucle la cual rellenaremos con botones siguiendo los siguientes pasos:

- Creamos el botón dentro del frame del tablero y asignándole el comando lambda con el método revelar_celda para cuando se haga clic

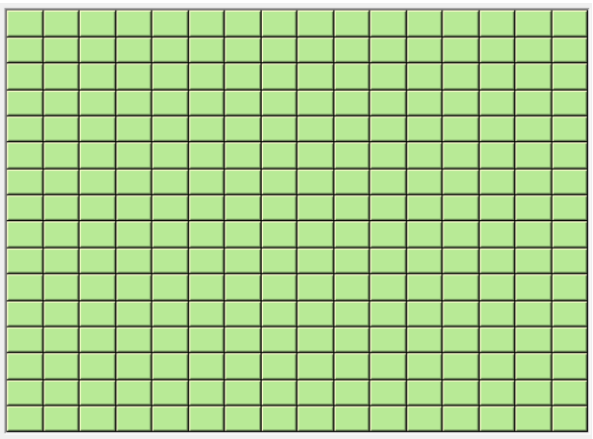
- Colocamos el botón en las posiciones x e y según el bucle y usamos la propiedad sticky=”nsew” para que se expandan en su cuadrícula del grid

- Cambiamos el color del fondo a verde

Excepcionalmente, si la dificultad es “Principiante”, el tamaño del botón es 5x2 para que ocupe el espacio necesario

Al final del segundo bucle se le añade a la lista de fila una nueva celda creada con las coordenadas “x” e “y” y el botón que acabamos de crear

Al final del primer bucle se añade la fila al parámetro “celdas”, consiguiendo así una lista bidimensional



► colocar_minas(self)

Este método se encarga de distribuir aleatoriamente las minas dentro del tablero

```
# Método para colocar las minas en el tablero
def colocar_minas(self):
    posiciones = random.sample([(x, y) for x in range(self.filas) for y in range(self.columnas)], self.minas)
    for x, y in posiciones:
        self.celdas[x][y].tiene_mina = True
```

- Se genera una lista de posiciones aleatorias con el eje "x" según el número de filas y el eje "y" según el número de columnas

- Se marca la propiedad tiene_mina como True en las celdas de las posiciones generadas.

► calcular_numeros_adyacentes(self)

```
# Método para calcular los números adyacentes a cada celda
def calcular_numeros_adyacentes(self):
    for x in range(self.filas):
        for y in range(self.columnas):
            if not self.celdas[x][y].tiene_mina:
                self.celdas[x][y].numero_adyacente = self.contar_minas_adyacentes(x, y)
```

Este método recorre el array de celdas y le aplica el método "contar_minas_adyacentes" a cada casilla sin mina

► contar_minas_adyacentes(self, x, y)

En este método tenemos una lista de direcciones, que serán las que comprobemos, rodeando la casilla que estamos tratando

```
# Método para contar las minas adyacentes a una celda
def contar_minas_adyacentes(self, x, y):
    direcciones = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]
    minas = 0
    for dx, dy in direcciones:
        nx, ny = x + dx, y + dy
        if 0 <= nx < self.filas and 0 <= ny < self.columnas and self.celdas[nx][ny].tiene_mina:
            minas += 1
    return minas
```

Inicializamos el contador de minas a 0 y comenzamos el bucle for con dos índices iterando en "direcciones", creamos variables dentro del bucle que serán las coordenadas que vamos a comprobar, nos aseguramos de respetar los límites del tablero y si existe mina en dicha coordenada, se le suma 1 al contador

Finalmente, se devuelve el número de minas

► iniciar_juego(self)

```python
# Método para iniciar el juego
def iniciar_juego(self):
    self.celdas = []
    self.minas_restantes = self.minas
    self.minas_restantes_label.config(text=str(self.minas_restantes))
    self.tiempo_inicio = None
    self.temporizador_activo = False
    self.temporizador.config(text="00:00")
    self.crear_tablero()
    self.colocar_minas()
    self.calcular_numeros_adyacentes()
```

Este método consiste en inicializar los valores necesarios de nuevo, ajustando valores como:

- Vaciar la lista de celdas

- Reiniciar el valor de las minas restantes a las minas actuales y actualizar también su label asociado

- Reiniciar el tiempo de inicio, indicar que el temporizador no está activo y reiniciar su label asociado

Finaliza ejecutando los últimos tres métodos explicados

► revelar_celda(self, x, y)

Este método recoge lo que ocurre al hacer clic a una casilla

```python
# Método para revelar una celda
def revelar_celda(self, x, y):
    if not self.temporizador_activo:
        self.tiempo_inicio = time.time()
        self.temporizador_activo = True
        self.actualizar_temporizador()

    celda = self.celdas[x][y]
    if celda.revelada or celda.marcada:
        return

    celda.revelada = True
    if celda.tiene_mina:
        self.game_over(perdio=True)
        return

    celda.boton.config(text=str(celda.numero_adyacente) if celda.numero_adyacente > 0 else "", bg="#a6ca8f", state="disabled")

    if celda.numero_adyacente == 0:
        self.revelar_en_cascada(x, y)

    self.verificar_victoria()
```

Comenzamos comprobando si el temporizador no está activo, esto lo hacemos para que el contador comience al primer clic en el tablero

Crea una variable celda tomando la que hemos pulsado mediante sus coordenadas

Después comprobamos si la celda que hemos pulsado estaba ya revelada o si está marcada con una bandera, si es así, terminamos el método sin hacer nada, si no fuera el caso, marcamos la celda como revelada y continuamos con el método

Comprobamos si la celda tiene mina, si tiene pierde el juego y finaliza el método, si no cambiamos la configuración de la celda:

- Cambiamos su texto al número de minas adyacentes si es mayor que 0

- Cambiamos el color de fondo a un verde más oscuro

- Cambiamos su estado a deshabilitado

Además, si el número de minas adyacentes de esta celda es 0, se ejecuta el método revelar en cascada

Por último, se comprueba si ha ganado

► revelar_en_cascada(self, x, y)

Este método controla el efecto de revelar el "espacio vacío" de casillas donde no hay minas adyacentes

```
# Método para revelar celdas en cascada
def revelar_en_cascada(self, x, y):
    direcciones = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]
    for dx, dy in direcciones:
        nx, ny = x + dx, y + dy
        if 0 <= nx < self.filas and 0 <= ny < self.columnas:
            self.revelar_celda(nx, ny)
```

Contamos con una lista de direcciones y un bucle for de dos índices iterando a "direcciones" igual al que usamos en el método "contar minas adyacentes" pero en este caso aplicamos el método revelar_celda a dicha celda adyacente

Este método, anidado con el anterior, crea un bucle que se detiene automáticamente cuando detecta un número diferente a 0, por lo que podemos revelar celdas "a lo loco" sin miedo a revelar una casilla con bomba, ya que nos encontraríamos un número antes

► marcar_celda(self, x, y)

Este método permite que podamos marcar una celda

Crea una variable celda tomando la que hemos pulsado mediante sus coordenadas

Comprobamos si la celda está revelada, por lo cual ya no tendría sentido marcarla, y finalizamos el método

```python
def marcar_celda(self, x, y):
    celda = self.celdas[x][y]
    if celda.revelada:
        return

    if celda.marcada:
        celda.boton.config(text="", bg="#b8ea96")
        self.minas_restantes += 1
        celda.marcada = not celda.marcada
    else:
        if self.minas_restantes > 0:
            celda.boton.config(text="\u2691", bg="lightblue")
            self.minas_restantes -= 1
            celda.marcada = not celda.marcada

    self.minas_restantes_label.config(text=str(self.minas_restantes))
```

Comprobamos también si la celda ya estaba marcada, si fuera así, configuramos el aspecto de la celda como al inicio, sumamos 1 a la variable minas_restantes que usamos también para las banderas restantes y cambiamos su estado a no marcada.

Por otro lado, si la celda no estaba marcada y la cantidad de minas_restantes es mayor a 0, configuramos el aspecto de la casilla para mostrar que está marcada, restamos 1 a la cantidad minas_restantes y establecemos que está marcada

Por ultimo, cambiamos el valor del label correspondiente a minas_restantes para reflejar los cambios

► verificar_victoria(self)

Este método comprueba si se ha ganado el juego

```python
# Método para verificar si se ha ganado el juego
def verificar_victoria(self):
    for fila in self.celdas:
        for celda in fila:
            if not celda.tiene_mina and not celda.revelada:
                return
    self.game_over(perdio=False)
```

Simplemente recorre todas las casillas y comprueba que todas las que tengan minas estén reveladas, si no fuera así, entraría al if y acabaría el método, si todas las casillas cumplen esta condición se ejecuta el método game_over indicando que no ha perdido, es decir, ha ganado

► actualizar_temporizador(self)

Con este método llevamos el control del temporizador

```python
# Método para actualizar el temporizador
def actualizar_temporizador(self):
    if self.temporizador_activo:
        tiempo_transcurrido = int(time.time() - self.tiempo_inicio)
        minutos = tiempo_transcurrido // 60
        segundos = tiempo_transcurrido % 60
        self.temporizador.config(text=f"{minutos:02}:{segundos:02}")
        self.ventana.after(1000, self.actualizar_temporizador)
```

Si el temporizador está activo, calcula el tiempo transcurrido restando el tiempo actual al de inicio

Convierte el tiempo transcurrido a minutos y segundos y actualizar el texto del temporizador en la interfaz según los valores obtenidos

Para que este método siga contando constantemente, se llama a si mismo nuevamente después de 1000 milisegundos (1 segundo)

► actualizar_records(self)

Este método se encarga de mostrar en la interfaz los records

```python
# Método para actualizar los récords en la interfaz
def actualizar_records(self):
    dificultad = self.dificultad_selector.get()
    records_text = "\n".join([f"{i+1}. {t//60:02}:{t%60:02}" for i, t in enumerate(self.records[dificultad][:5])]
    self.record_puntos.config(text=records_text)
```

Según la dificultad que esté seleccionada, recoge los 5 mejores records de dicha dificultad y los concatena de modo que quede con un formato así:

Para que se muestre en nuestro programa, establecemos el texto del label correspondiente a los records este texto generado

► cambiar_dificultad(self)

Este método ajusta los valores de filas, columnas y minas dependiendo de la dificultad que se haya seleccionado

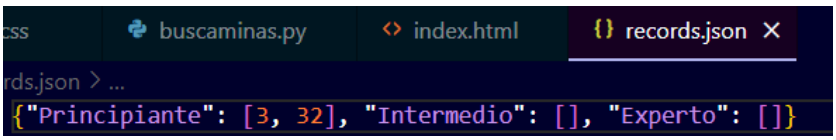
Después, ejecuta los métodos iniciar_juego y actualizar_records

```python
# Método para cambiar la dificultad del juego
def cambiar_dificultad(self):
    dificultad = self.dificultad_selector.get()
    if dificultad == "Principiante":
        self.filas = 9; self.columnas = 9; self.minas = 10
    elif dificultad == "Intermedio":
        self.filas = 16; self.columnas = 16; self.minas = 40
    elif dificultad == "Experto":
        self.filas = 16; self.columnas = 30; self.minas = 99

    self.iniciar_juego()
    self.actualizar_records()
```

► guardar_records(self)

```python
# Método para guardar los récords en un archivo
def guardar_records(self):
    with open("records.json", "w") as file:
        json.dump(self.records, file)
```

Para poder recuperar los records anteriores después de cerrar el programa, los guardaremos en un archivo .json, con este métodos guardamos los datos en el diccionario "records" a nuestro archivo, que tiene la misma estructura



► cargar_records(self)

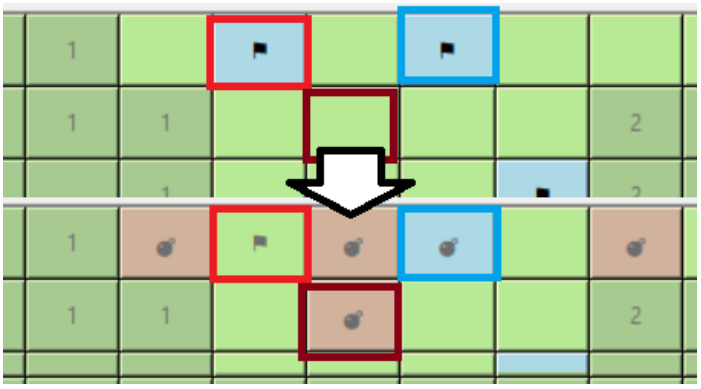
Este método complementa al anterior y es que con este cargamos los records almacenados en el archivo .json para poder ser visibles en un nuevo juego

```python
# Método para cargar los récords desde un archivo
def cargar_records(self):
    if os.path.exists("records.json"):
        try:
            with open("records.json", "r") as file:
                self.records = json.load(file)
        except json.JSONDecodeError:
            self.records = {"Principiante": [], "Intermedio": [], "Experto": []}
```

► game_over(self, perdio)

Por último, este método controla si pierde o si gana mediante el parámetro "perdio"

Primero cambia el aspecto del tablero para mostrar todas las minas, las que estaban marcadas mantienen su color de fondo y las que no cambia su fondo a marrón



Además, las casillas marcadas sin bombas mantienen el icono de la bandera y el color cambia al mismo verde de las casillas al inicio

Esto genera este efecto:

Por último en el bucle, se deshabilita la celda, acabando con todas deshabilitadas

Después, se deshabilita la actividad del temporizador

Si no perdió, es decir, ganó, se calcula el tiempo transcurrido restando el tiempo actual y el tiempo de inicio y tomando la dificultad según su valor en la lista desplegable, se guarda en su lista correspondiente dentro del diccionario de records y se ordena dicha lista. Después se ejecutan los métodos actualizar_records y guardar_records

Finalmente, se muestra un mensaje al usuario que indicará si ha ganado o perdido seguido de la pregunta "¿Quieres jugar de nuevo?", si el usuario pulsa "Sí" ejecutamos el método cambiar_dificultad, reiniciando el juego, si pulsa "No" acabamos el programa destruyendo la ventana principal

```
# Método para manejar el fin del juego
def game_over(self, perdio):
    for fila in self.celdas:
        for celda in fila:
            if celda.tiene_mina:
                if celda.marcada:
                    celda.boton.config(text="💣", state="disabled", bg="lightblue")
                else:
                    celda.boton.config(text="💣", state="disabled", bg="#d1b29c")
            else:
                if celda.marcada:
                    celda.boton.config(bg="#b8ea96")
                celda.boton.config(state="disabled")

    self.temporizador_activo = False

    if not perdio:
        tiempo_transcurrido = int(time.time() - self.tiempo_inicio)
        dificultad = self.dificultad_selector.get()
        self.records[dificultad].append(tiempo_transcurrido)
        self.records[dificultad].sort()  # Ordenar los récords de menor a mayor
        self.actualizar_records()
        self.guardar_records()  # Guardar los récords en el archivo

    mensaje = "Has perdido" if perdio else "Has ganado"
    respuesta = messagebox.askyesno("Fin del juego", f"{mensaje}. ¿Quieres jugar de nuevo?")
    if respuesta:
        self.cambiar_dificultad()
    else:
        self.ventana.destroy()
```

Ejecución del código

Con este código creamos una nueva clase Buscaminas que se ejecuta automáticamente al iniciar

```
if __name__ == "__main__":
    app = Buscaminas()  # I
    app.ventana.mainloop()
```

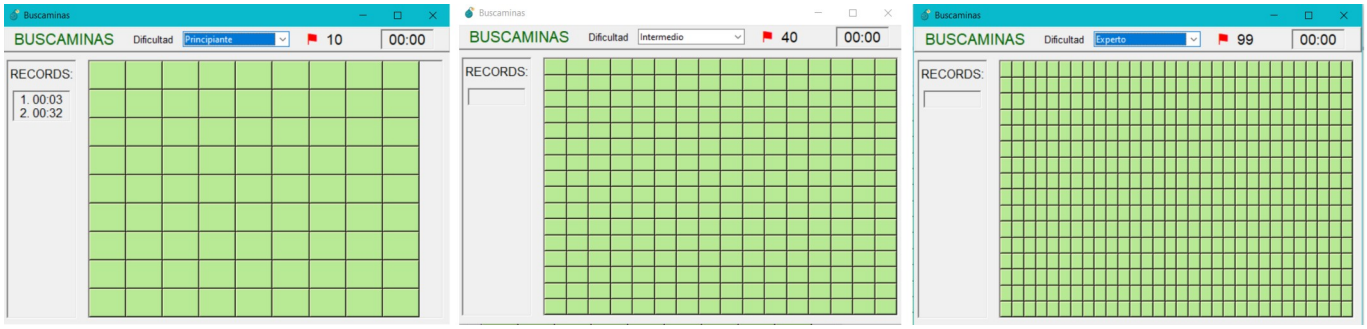
Apariencia visual

En este apartado vamos a recalcar algunos de los visuales que hemos conseguido con Tkinter

Empezamos con el tablero dependiendo de su dificultad, vemos como las casillas cambian de tamaño y de cantidad, pero siempre almacenadas en el mismo *frame* sin rebasar

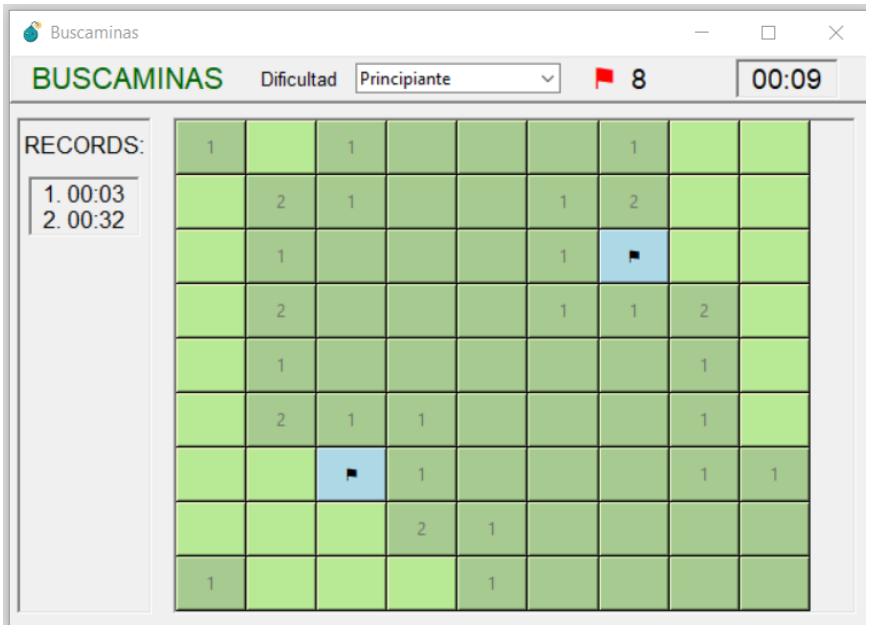
Aquí también podemos apreciar como aparecen los records que tenemos almacenados, como no tenemos ninguno en las dificultades “Intermedio” y “Dificil”, está vacío

Además, apreciamos como el número de banderas ha cambiado, correspondiendo al número de minas que hay en el tablero



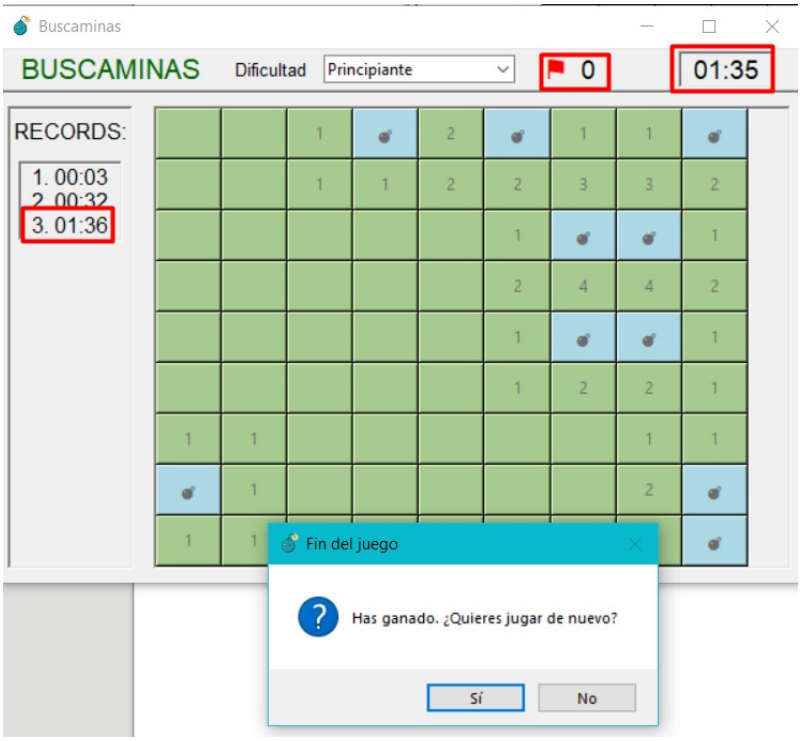
Una vez ejecutamos el juego, podemos encontrarnos con los “espacios vacíos” mencionados anteriormente, aquí hay podemos ver como hemos conseguido revelar en cascada ese espacio cambiado su color

Además, podemos ver como el temporizador ha avanzado y como el número de banderas ha disminuido a 8, porque hemos colocado 2 banderas



Cuando finalizamos el juego y ganamos, nuestra puntuación se almacena automáticamente en el archivo .json y se muestra en el label de los records

Podemos comprobar como el número de banderas efectivamente es igual al número de minas en el tablero



Aspectos innovadores

La interfaz y el código de colores usado hace que la experiencia del jugador sea sencilla e intuitiva

Conclusión y opinión personal

En conclusión, este proyecto implementa funcionalidades de Tkinter no muy complicadas con lo que hemos podido crear un juego completo, demostrando así que muchas cosas simples pueden llevar a algo muy complejo

Además, comparado con proyectos anteriores, es mucho más reconfortante ver como tu trabajo toma forma fuera de una consola, pudiendo personalizarlo tanto como queramos

Si bien la creación de este proyecto se ha visto muy de la mano de asistentes IA, debido a que comenzar con una librería nueva sin conocerla es un trabajo muy tedioso, no hay nada que esté fuera de mi comprensión, lo que me hace deducir que Tkinter es una librería intuitiva y sencilla y no tendría ningún problema en usarla de nuevo para otro proyecto en un futuro

Bibliografía

Vídeos proporcionados por el profesor

- https://www.youtube.com/watch?v=U1htWuCwgDg&ab_channel=ManuelGonz%C3%A1lez

- https://www.youtube.com/watch?v=XK2bG7fMBms&ab_channel=ManuelGonz%C3%A1lez

- https://www.youtube.com/watch?v=O2mlGuaVTxY&ab_channel=ManuelGonz%C3%A1lez

- https://www.youtube.com/watch?v=NU3MGDvNNnI&ab_channel=ManuelGonz%C3%A1lez

Consultas a Asistentes de IA

- Recrear interfaz según imagen

- Genera un buscaminas básico *(Tomado de ejemplo e implementado algunas partes)*

- Teniendo en cuenta este codigo, mejora la interfaz para que el tablero de juego ocupe siempre el tamaño de su contenedor dependiendo de la dificultad establecida

- haz funcional el temporizador, que empiece a contar cuando haga el primer clic a una casilla hasta que pierda o gane, si gana, se almacenará en una lista de records

- Explicar estructura necesaria para el archivo JSON

- Como implementar la lectura y escritura del archivo JSON