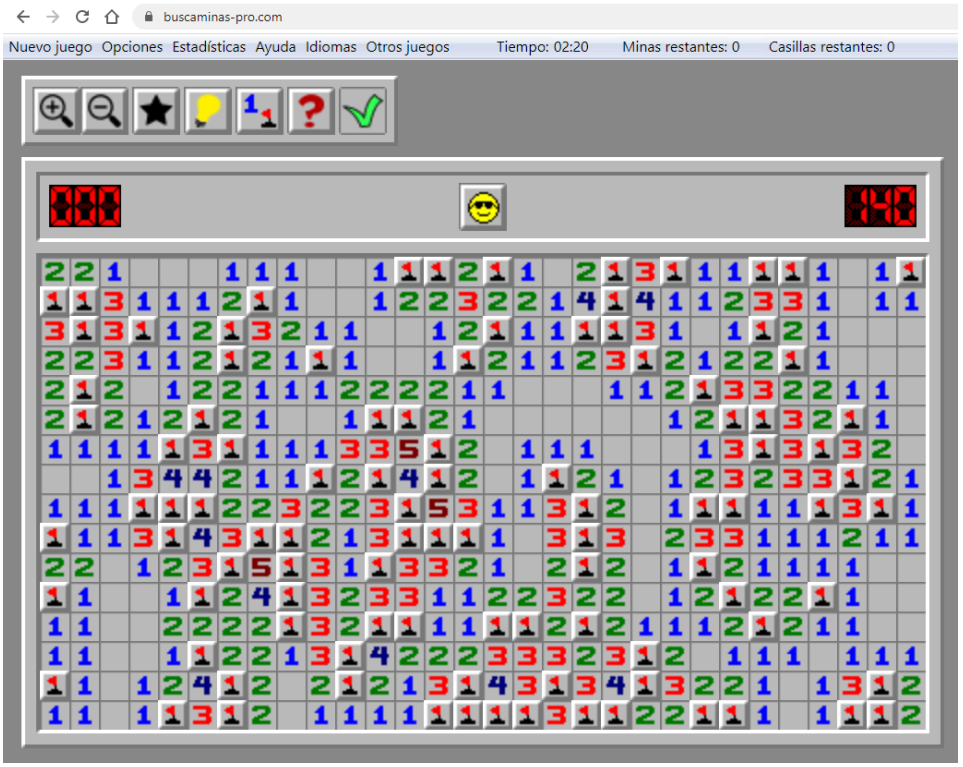

PYTHON BUSCAMINAS

UT - 3. 1. PYTHON BUSCAMINAS

Apellidos: Núñez García

Nombre: Héctor

Nº PC: 05

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: 12/02/2025

Resumen del Proyecto: Buscaminas en Python con Tkinter

1. Estructura del código

La estructura del código está organizada orientada a objetos. Esto permite que el manejo de las funciones y diferentes partes del programa sean más fáciles de manejar. Se basa en una gran clase:

Clase Principal (Buscaminas) : Contiene toda la funcionalidad del juego.

- **Atributos** :
  - **root :**
    - Tipo: Objeto Tkinter.Tk
    - Descripción: Representa la ventana principal de la aplicación.
    - Uso: Se utiliza para gestionar todos los widgets y eventos de la interfaz gráfica.
  - **dificultad_seleccionada :**
    - Tipo: String
    - Descripción: Almacena la dificultad seleccionada por el usuario ("Facil", "Medio" o "Dificil").
    - Uso: Determina las dimensiones del tablero y el número de minas.
  - **tablero :**
    - Tipo: Lista de listas (matriz)
    - Descripción: Representa el estado interno del tablero, incluyendo información sobre minas, adyacencias, revelado y banderas.
    - Estructura: Cada celda es un diccionario con las siguientes claves:
    - "mina": Booleano que indica si hay una mina en la celda.
    - "adyacentes": Entero que indica cuántas minas están cerca.
    - "revelado": Booleano que indica si la celda ha sido revelada.
    - "bandera": Booleano que indica si la celda está marcada con una bandera.
  - **botones :**
    - Tipo: Lista de listas (matriz)
    - Descripción: Contiene los botones visuales que representan las celdas del tablero.
    - Uso: Permite interactuar con el tablero mediante clics izquierdos y derechos.
  - **inicio_tiempo :**
    - Tipo: Float
    - Descripción: Marca el tiempo en segundos cuando comienza el juego.
    - Uso: Se utiliza para calcular el tiempo transcurrido durante la partida.
  - **tiempo_label :**
    - Tipo: Objeto Tkinter.Label
    - Descripción: Muestra el tiempo transcurrido en la interfaz gráfica.
    - Uso: Se actualiza periódicamente mediante el método actualizar_tiempo().
  - **banderas_usadas :**
    - Tipo: Entero
    - Descripción: Indica cuántas banderas ha colocado el jugador.
    - Uso: Limita el uso de banderas al número total de minas.
  - **banderas_totales :**
    - Tipo: Entero
    - Descripción: Indica el número total de banderas disponibles (igual al número de minas).

- Uso: Se compara con banderas_usadas para evitar que el jugador use más banderas de las permitidas.

- **banderas_label :**
  - Tipo: Objeto Tkinter.Label
  - Descripción: Muestra el número de banderas usadas y disponibles.
  - Uso: Se actualiza automáticamente cuando el jugador coloca o quita una bandera.
- **records :**
  - Tipo: Diccionario
  - Descripción: Almacena los mejores tiempos registrados para cada dificultad.
  - Estructura: Las claves son las dificultades ("Fácil", "Medio", "Difícil") y los valores son los tiempos en segundos.
- **combobox_dificultad :**
  - Tipo: Objeto Tkinter.ttk.Combobox
  - Descripción: Permite al usuario seleccionar la dificultad del juego.
  - Uso: Su valor seleccionado se almacena en dificultad_seleccionada.

- **Métodos :**
  - **__init__(self, root) :**
    - Propósito: Constructor de la clase. Configura la ventana principal y muestra el menú inicial.
    - Acciones:
      - Establece el título de la ventana.
      - Inicializa los atributos de la clase.
      - Carga los archivos de sonidos
      - Llama a crear_menu_dificultad() para mostrar el menú principal.

```python
class Buscaminas:
    def __init__(self, root):
        self.root = root
        self.root.title("Buscaminas")  # Título de la ventana principal
        self.dificultad_seleccionada = None  # Dificultad seleccionada por el usuario
        self.tablero = None  # Representación interna del tablero
        self.botones = []  # Lista de botones que representan el tablero visual
        self.inicio_tiempo = None  # Marca el inicio del temporizador
        self.tiempo_label = None  # Etiqueta que muestra el tiempo transcurrido
        self.banderas_usadas = 0  # Número de banderas colocadas
        self.banderas_totales = 0  # Número total de banderas disponibles
        self.banderas_label = None  # Etiqueta que muestra el estado de las banderas
        self.records = self.cargar_records()  # Carga los récords desde el archivo
        self.crear_menu_dificultad()  # Muestra el menú de selección de dificultad

        # Inicializar pygame para manejar el sonido
        pygame.mixer.init()

        # Cargar el archivo de sonido para cuando se pulse una mina
        try:
            self.sonido_mina = pygame.mixer.Sound("sonidos/peo.mp3")
            self.sonido_bandera = pygame.mixer.Sound("sonidos/chiclin.mp3")
        except FileNotFoundError:
            print("Error: No se encontró el archivo de sonido 'peo.mp3'.")
            self.sonido_mina = None
```

  - **crear_menu_dificultad(self) :**
    - Propósito: Crea y muestra el menú inicial donde el usuario selecciona la dificultad.
    - Acciones:

- Limpia la ventana actual.
- Crea un Frame principal con un título, un Combobox para seleccionar la dificultad y varios botones (Jugar, Récords, Salir).

```python
# Crea el menú inicial donde el usuario selecciona la dificultad
def crear_menu_dificultad(self):
    """Crea el menú de inicio con un título, un Combobox y botones estilizados."""
    # Limpiar la ventana actual
    for widget in self.root.winfo_children():
        widget.destroy()

    # Frame principal
    frame = tk.Frame(self.root, bg="#f0f0f0")
    frame.pack(expand=True, fill="both", padx=20, pady=20)

    self.root.geometry("400x400")  # Tamaño del menú inicial

    # Título del menú
    titulo = tk.Label(frame, text="Bienvenido al Buscaminas\nmasón de Héctor", font=("Arial", 20, "bold"), bg="#f0f0f0", fg="#333333")
    titulo.pack(pady=20)

    # Etiqueta para el Combobox
    label = tk.Label(frame, text="Selecciona la dificultad:", font=("Arial", 14), bg="#f0f0f0", fg="#333333")
    label.pack(pady=10)

    # Combobox para seleccionar la dificultad
    self.combobox_dificultad = ttk.Combobox(frame, values=list(DIFICULTADES.keys()), state="readonly", font=("Arial", 12))
    self.combobox_dificultad.set("Facil")  # Valor predeterminado
    self.combobox_dificultad.pack(pady=10)

    # Botón para jugar
    boton_jugar = tk.Button(frame, text="Jugar", font=("Arial", 14), bg="#4CAF50", fg="white", command=self.iniciar_juego)
    boton_jugar.pack(pady=10)

    # Botón para ver los récords
    boton_records = tk.Button(frame, text="Récords", font=("Arial", 14), bg="#2196F3", fg="white", command=self.mostrar_ventana_records)
    boton_records.pack(pady=10)

    # Botón para salir
    boton_salir = tk.Button(frame, text="Salir", font=("Arial", 14), bg="#f44336", fg="white", command=self.root.quit)
    boton_salir.pack(pady=10)
```

Métodos de Gestión de Récords

- **cargar_records(self) :**
  - Propósito: Lee los récords almacenados en el archivo JSON.
  - Acciones:
    - Intenta abrir el archivo RECORDS_FILE.
    - Si el archivo no existe, devuelve un diccionario con valores predeterminados (None para todas las dificultades).

```python
# Carga los récords desde el archivo JSON
def cargar_records(self):
    """Carga los récords desde el archivo JSON."""
    try:
        with open(RECORDS_FILE, "r") as file:
            return json.load(file)
    except FileNotFoundError:
        return {"Fácil": None, "Medio": None, "Difícil": None}
```

- **guardar_records(self) :**
  - Propósito: Guarda los récords actuales en el archivo JSON.
  - Acciones:
    - Escribe el contenido del atributo records en el archivo RECORDS_FILE.

```python
def guardar_records(self):
    """Guarda los récords en el archivo JSON."""
    with open(RECORDS_FILE, "w") as file:
        json.dump(self.records, file)
```

- **guardar_record(self, tiempo) :**
  - Propósito: Comprueba si el tiempo proporcionado es un nuevo récord para la dificultad actual y lo guarda si es necesario.
  - Acciones:
    - Compara el tiempo proporcionado con el récord existente.
    - Si es mejor, actualiza el récord y muestra un mensaje emergente.

```python
# Guarda los récords en el archivo JSON
def guardar_record(self, tiempo):
    """Guarda el récord si es el mejor tiempo para la dificultad actual."""
    dificultad = self.combobox_dificultad.get()
    if self.records[dificultad] is None or tiempo < self.records[dificultad]:
        self.records[dificultad] = tiempo
        self.guardar_records()  # Llama al método correcto
        messagebox.showinfo("Récord", f"¡Nuevo récord para {dificultad}: {tiempo} segundos!")
```

Métodos de Generación del Tablero

- **generar_tablero(self) :**
  - Propósito: Crea una matriz vacía para representar el tablero.
  - Acciones:
    - Inicializa tablero como una lista de listas, donde cada celda es un diccionario con valores predeterminados (mina=False, adyacentes=0, revelado=False, bandera=False).

```python
# Genera una matriz vacía para representar el tablero
def generar_tablero(self):
    """Genera una matriz para representar el tablero."""
    self.tablero = [[{"mina": False, "adyacentes": 0, "revelado": False, "bandera": False} for _ in range(self.columnas)] for _ in range(self.filas)]
```

- **colocar_minas(self) :**
  - Propósito: Coloca las minas aleatoriamente en el tablero.
  - Acciones:
    - Utiliza un bucle while para colocar el número exacto de minas definido por la dificultad.
    - Asegura que no se coloquen minas en celdas ya ocupadas.

```python
# Coloca las minas aleatoriamente en el tablero
def colocar_minas(self):
    """Coloca las minas aleatoriamente en el tablero."""
    minas_colocadas = 0
    while minas_colocadas < self.minas:
        fila = random.randint(0, self.filas - 1)
        columna = random.randint(0, self.columnasD - 1)
        if not self.tablero[fila][columna]["mina"]:
            self.tablero[fila][columna]["mina"] = True
            minas_colocadas += 1
```

- **calcular_adyacencias(self) :**
  - Propósito: Calcula el número de minas adyacentes para cada celda.
  - Acciones:
    - Itera sobre todas las celdas del tablero.
    - Si una celda no contiene una mina, llama a contar_minas_adyacentes() para determinar cuántas minas hay cerca.

```python
# Calcula el número de minas adyacentes para cada celda
def calcular_adyacencias(self):
    """Calcula el número de minas adyacentes para cada celda."""
    for fila in range(self.filas):
        for columna in range(self.columnas):
            if not self.tablero[fila][columna]["mina"]:
                self.tablero[fila][columna]["adyacentes"] = self.contar_minas_adyacentes(fila, columna)
```

- **contar_minas_adyacentes(self, fila, columna) :**
  - Propósito: Cuenta las minas en las ocho posiciones circundantes de una celda específica.
  - Acciones:
    - Itera sobre las filas y columnas adyacentes.
    - Verifica que las coordenadas estén dentro de los límites del tablero y cuenta las minas encontradas.

```python
# Cuenta las minas adyacentes a una celda específica
def contar_minas_adyacentes(self, fila, columna):
    """Cuenta las minas adyacentes a una celda específica."""
    minas = 0
    for df in [-1, 0, 1]:
        for dc in [-1, 0, 1]:
            if df == 0 and dc == 0:
                continue
            f, c = fila + df, columna + dc
            if 0 <= f < self.filas and 0 <= c < self.columnas and self.tablero[f][c]["mina"]:
                minas += 1
    return minas
```

Métodos de Interacción con el Jugador

- **iniciar_juego(self) :**
  - Propósito: Inicia el juego con la dificultad seleccionada.
  - Acciones:
    - Obtiene la configuración correspondiente a la dificultad seleccionada.
    - Limpia la ventana actual y ajusta su tamaño según las dimensiones del tablero.
    - Genera el tablero, coloca las minas y calcula las adyacencias.
    - Crea la interfaz gráfica del tablero y comienza el temporizador.

```python
# Inicia el juego con la dificultad seleccionada
def iniciar_juego(self):
    """Inicia el juego con la dificultad seleccionada."""
    dificultad = self.combobox_dificultad.get()
    if not dificultad:
        messagebox.showwarning("Advertencia", "Por favor, selecciona una dificultad.")
        return

    # Almacenar la dificultad seleccionada
    self.dificultad_seleccionada = dificultad

    # Obtener configuración de la dificultad
    config = DIFICULTADES[dificultad]
    self.filas, self.columnas, self.minas = config["filas"], config["columnas"], config["minas"]
    self.banderas_totales = self.minas
    self.banderas_usadas = 0

    # Limpiar la ventana actual
    for widget in self.root.winfo_children():
        widget.destroy()

    # Ajustar el tamaño de la ventana según la dificultad
    self.root.geometry(f"{self.columnas * 35}x{self.filas * 35 + 150}")

    # Inicializar variables del juego
    self.generar_tablero()
    self.colocar_minas()
    self.calcular_adyacencias()

    # Crear el tablero gráfico
    self.crear_tablero()

    # Contador de tiempo
    self.inicio_tiempo = time.time()
    self.actualizar_tiempo()
```

- **crear_tablero(self) :**
  - Propósito: Crea la interfaz gráfica del tablero.
  - Acciones:
    - Crea un Frame principal y organiza los botones en una cuadrícula.
    - Asigna eventos a los botones para manejar clics izquierdos (revelar_celda) y derechos (colocar_bandera).
    - Agrega etiquetas para mostrar el tiempo transcurrido y el estado de las banderas.

```python
# Crea la interfaz gráfica del tablero
def crear_tablero(self):
    """Crea la interfaz gráfica del tablero."""
    frame = tk.Frame(self.root, bg="#2c3e50")
    frame.pack(pady=10)

    self.botones = []
    for fila in range(self.filas):
        fila_botones = []
        for columna in range(self.columnas):
            boton = tk.Button(frame, width=2, height=1,
                              bg="#e0e0e0", relief="raised", bd=2, font=("Arial", 10, "bold"))
            boton.grid(row=fila, column=columna)
            boton.bind("<Button-1>", lambda event, f=fila, c=columna: self.revelar_celda(f, c))  # Clic izquierdo
            boton.bind("<Button-3>", lambda event, f=fila, c=columna: self.colocar_bandera(f, c))  # Clic derecho
            fila_botones.append(boton)
        self.botones.append(fila_botones)

    # Etiqueta de tiempo
    self.tiempo_label = tk.Label(self.root, text="Tiempo: 0 segundos", font=("Arial", 12), bg="#2c3e50", fg="#ecf0f1")
    self.tiempo_label.pack(pady=10)

    # Etiqueta de banderas
    self.banderas_label = tk.Label(self.root, text=f"Banderas: {self.banderas_usadas}/{self.banderas_totales}", font=("Arial", 12), bg="#2c3e50", fg="#ecf0f1")
    self.banderas_label.pack(pady=10)
```

- **revelar_celda(self, fila, columna) :**
  - Propósito: Maneja el clic izquierdo en una celda.
  - Acciones:
    - Si la celda contiene una mina, termina el juego mostrando un mensaje de derrota.
    - Si no hay minas, revela el número de minas adyacentes o, si no hay ninguna, revela recursivamente las celdas vecinas.
    - Comprueba si el jugador ha ganado.

```python
# Revela el contenido de una celda
def revelar_celda(self, fila, columna):
    """Revela el contenido de una celda."""
    celda = self.tablero[fila][columna]

    if celda["bandera"] or celda["revelado"]:
        return

    celda["revelado"] = True
    boton = self.botones[fila][columna]

    if celda["mina"]:
        boton.config(text="💣", bg="#e74c3c", relief="sunken", state="disabled")
        self.destapar_tablero_perdido()  # Destapar el tablero antes de mostrar el mensaje
        self.mostrar_mensaje_final("¡Has perdido!")
    else:
        color_texto = COLORES_FONDO.get(celda["adyacentes"], "black")  # Color del texto
        color_fondo = COLORES_FONDO.get(celda["adyacentes"], "#b0b0b0")  # Color del fondo
        texto = str(celda["adyacentes"]) if celda["adyacentes"] > 0 else ""
        boton.config(text=texto, bg=color_fondo, fg=color_texto, relief="sunken", state="disabled")
        if celda["adyacentes"] == 0:
            self.revelar_celdas_adyacentes(fila, columna)

    if self.verificar_victoria():
        tiempo = int(time.time() - self.inicio_tiempo)
        self.guardar_record(tiempo)
        self.mostrar_mensaje_final("¡Has ganado!")
```

- **revelar_celdas_adyacentes(self, fila, columna) :**
  - Propósito: Revela recursivamente las celdas adyacentes sin minas.
  - Acciones:
    - Itera sobre las filas y columnas adyacentes.
    - Llama a revelar_celda() para cada celda válida.

```python
# Revela recursivamente las celdas adyacentes sin minas
def revelar_celdas_adyacentes(self, fila, columna):
    """Revela recursivamente las celdas adyacentes si no hay minas cercanas."""
    for df in [-1, 0, 1]:
        for dc in [-1, 0, 1]:
            if df == 0 and dc == 0:
                continue
            f, c = fila + df, columna + dc
            if 0 <= f < self.filas and 0 <= c < self.columnas:
                self.revelar_celda(f, c)
```

- **colocar_bandera(self, fila, columna) :**
  - Propósito: Maneja el clic derecho en una celda para colocar o quitar una bandera.
  - Acciones:
    - Cambia el estado de la bandera en la celda seleccionada.
    - Actualiza el aspecto visual del botón correspondiente.
    - Actualiza el contador de banderas en la interfaz gráfica.

```python
# Coloca o quita una bandera en una celda
def colocar_bandera(self, fila, columna):
    """Coloca o quita una bandera en una celda."""
    celda = self.tablero[fila][columna]
    boton = self.botones[fila][columna]

    if celda["revelado"]:
        return

    if celda["bandera"]:
        # Quitar bandera
        celda["bandera"] = False
        boton.config(text="", bg="#e0e0e0", relief="raised", state="normal")
        self.banderas_usadas -= 1
    else:
        if self.banderas_usadas < self.banderas_totales:
            # Colocar bandera
            celda["bandera"] = True
            boton.config(text="🚩", bg="#f1c40f", relief="raised", state="normal")
            self.banderas_usadas += 1

    # Actualizar etiqueta de banderas
    self.banderas_label.config(text=f"Banderas: {self.banderas_usadas}/{self.banderas_totales}")
```

Métodos de Finalización del Juego

- **mostrar_mensaje_final(self, mensaje) :**
  - Propósito: Muestra un mensaje emergente al final del juego.
  - Acciones:
    - Detiene el temporizador.
    - Muestra un mensaje indicando si el jugador ha ganado o perdido.
    - Llama a volver_al_menu() para reiniciar el programa.

```python
# Muestra un mensaje al final del juego
def mostrar_mensaje_final(self, mensaje):
    """Muestra un mensaje al final del juego."""
    self.inicio_tiempo = None  # Detener el contador de tiempo
    messagebox.showinfo("Fin del juego", mensaje)
    self.volver_al_menu()
```

- **destapar_tablero_perdido(self) :**
  - Propósito: Revela todo el tablero cuando el jugador pierde.
  - Acciones:
    - Muestra todas las minas con un ícono especial (💣).
    - Revela el número de minas adyacentes en las celdas restantes.

```python
# Destapa todo el tablero cuando el jugador pierde
def destapar_tablero_perdido(self):
    """Destapa todo el tablero cuando el jugador pierde."""
    for fila in range(self.filas):
        for columna in range(self.columnas):
            celda = self.tablero[fila][columna]
            boton = self.botones[fila][columna]

            if celda["mina"]:
                boton.config(text="💣", bg="#e74c3c", relief="sunken", state="disabled")
            elif celda["adyacentes"] > 0:
                color_texto = COLORES_FONDO.get(celda["adyacentes"], "black")
                color_fondo = COLORES_FONDO.get(celda["adyacentes"], "#b0b0b0")
                texto = str(celda["adyacentes"])
                boton.config(text=texto, bg=color_fondo, fg=color_texto, relief="sunken", state="disabled")
            else:
                boton.config(bg="#b0b0b0", relief="sunken", state="disabled")
```

- **verificar_victoria(self) :**
  - Propósito: Comprueba si el jugador ha ganado.
  - Acciones:
    - Itera sobre todas las celdas del tablero.
    - Devuelve True si todas las celdas sin minas han sido reveladas.

```python
# Verifica si el jugador ha ganado
def verificar_victoria(self):
    """Verifica si el jugador ha ganado."""
    for fila in range(self.filas):
        for columna in range(self.columnas):
            celda = self.tablero[fila][columna]
            if not celda["mina"] and not celda["revelado"]:
                return False
    return True
```

- **volver_al_menu(self) :**
  - Propósito: Vuelve al menú principal después de completar una partida.
  - Acciones:
    - Limpia la ventana actual.
    - Llama a crear_menu_dificultad() para mostrar el menú inicial.

```python
def volver_al_menu(self):
    """Vuelve al menú principal."""
    for widget in self.root.winfo_children():
        widget.destroy()
    self.crear_menu_dificultad()
```

Métodos de Actualización

- **actualizar_tiempo(self) :**
  - Propósito: Actualiza el contador de tiempo en la interfaz gráfica.

- Acciones:
  - Calcula el tiempo transcurrido desde inicio_tiempo.
  - Actualiza el texto de tiempo_label.
  - Programa la llamada recursiva para actualizar el tiempo cada segundo.

```python
# Actualiza el contador de tiempo cada segundo
def actualizar_tiempo(self):
    """Actualiza el contador de tiempo."""
    if self.inicio_tiempo:
        tiempo_transcurrido = int(time.time() - self.inicio_tiempo)
        self.tiempo_label.config(text=f"Tiempo: {tiempo_transcurrido} segundos")
        self.root.after(1000, self.actualizar_tiempo)
```

Parte menú:

El menú principal se organiza dentro del método **crear_menu_dificultad()**, que se llama automáticamente al iniciar la aplicación en el constructor **(__init__)**. **Este método configura los siguientes elementos:**

- **Título del Juego :** Un texto descriptivo que da la bienvenida al jugador.
- **Selección de Dificultad :** Un Combobox (desplegable) que permite al usuario elegir entre las opciones disponibles: "Fácil", "Medio" o "Difícil".
- **Botones de Acción :**
  - **Jugar :** Inicia el juego con la dificultad seleccionada.
  - **Récords :** Muestra una ventana con los mejores tiempos registrados.
  - **Salir :** Cierra la aplicación.

```python
# Crea el menú inicial donde el usuario selecciona la dificultad
def crear_menu_dificultad(self):
    """Crea el menú de inicio con un título, un Combobox y botones estilizados."""
    # Limpiar la ventana actual
    for widget in self.root.winfo_children():
        widget.destroy()

    # Frame principal
    frame = tk.Frame(self.root, bg="#f0f0f0")
    frame.pack(expand=True, fill="both", padx=20, pady=20)

    self.root.geometry("400x400")  # Tamaño del menú inicial

    # Título del menú
    titulo = tk.Label(frame, text="Bienvenido al Buscaminas\nmasón de Héctor", font=("Arial", 20, "bold"), bg="#f0f0f0", fg="#333333")
    titulo.pack(pady=20)

    # Etiqueta para el Combobox
    label = tk.Label(frame, text="Selecciona la dificultad:", font=("Arial", 14), bg="#f0f0f0", fg="#333333")
    label.pack(pady=10)

    # Combobox para seleccionar la dificultad
    self.combobox_dificultad = ttk.Combobox(frame, values=list(DIFICULTADES.keys()), state="readonly", font=("Arial", 12))
    self.combobox_dificultad.set("Facil")  # Valor predeterminado
    self.combobox_dificultad.pack(pady=10)

    # Botón para jugar
    boton_jugar = tk.Button(frame, text="Jugar", font=("Arial", 14), bg="#4CAF50", fg="white", command=self.iniciar_juego)
    boton_jugar.pack(pady=10)

    # Botón para ver los récords
    boton_records = tk.Button(frame, text="Récords", font=("Arial", 14), bg="#2196F3", fg="white", command=self.mostrar_ventana_records)
    boton_records.pack(pady=10)

    # Botón para salir
    boton_salir = tk.Button(frame, text="Salir", font=("Arial", 14), bg="#f44336", fg="white", command=self.root.quit)
    boton_salir.pack(pady=10)
```

Además del método principal crear_menu_dificultad(), **otras funciones complementan la funcionalidad del menú:**

- **iniciar_juego() :** Se llama cuando el usuario selecciona la opción "Jugar". Verifica que se haya seleccionado una dificultad válida y configura el tablero del juego.

```python
# Inicia el juego con la dificultad seleccionada
def iniciar_juego(self):
    """Inicia el juego con la dificultad seleccionada."""
    dificultad = self.combobox_dificultad.get()
    if not dificultad:
        messagebox.showwarning("Advertencia", "Por favor, selecciona una dificultad.")
        return

    # Almacenar la dificultad seleccionada
    self.dificultad_seleccionada = dificultad

    # Obtener configuración de la dificultad
    config = DIFICULTADES[dificultad]
    self.filas, self.columnas, self.minas = config["filas"], config["columnas"], config["minas"]
    self.banderas_totales = self.minas
    self.banderas_usadas = 0

    # Limpiar la ventana actual
    for widget in self.root.winfo_children():
        widget.destroy()

    # Ajustar el tamaño de la ventana según la dificultad
    self.root.geometry(f"{self.columnas * 35}x{self.filas * 35 + 150}")

    # Inicializar variables del juego
    self.generar_tablero()
    self.colocar_minas()
    self.calcular_adyacencias()

    # Crear el tablero gráfico
    self.crear_tablero()

    # Contador de tiempo
    self.inicio_tiempo = time.time()
    self.actualizar_tiempo()
```

- **Mostrar_ventana_records() :** Muestra una ventana emergente con los récords actuales para cada nivel de dificultad.

```python
# Muestra una ventana con los récords actuales
def mostrar_ventana_records(self):
    """Muestra una ventana con la tabla de récords."""
    # Crear una nueva ventana
    ventana_records = tk.Toplevel(self.root)
    ventana_records.title("Tabla de Récords")
    ventana_records.geometry("400x300")
    ventana_records.resizable(False, False)

    # Frame principal
    frame = tk.Frame(ventana_records, bg="#f0f0f0")
    frame.pack(expand=True, fill="both", padx=20, pady=20)

    # Título de la ventana
    titulo = tk.Label(frame, text="Tabla de Récords", font=("Arial", 18, "bold"), bg="#f0f0f0", fg="#333333")
    titulo.grid(row=0, column=0, columnspan=2, pady=10)  # Usar grid para el título

    # Mostrar los récords en una tabla
    for i, dificultad in enumerate(DIFICULTADES.keys()):
        tiempo = self.records[dificultad]
        tiempo_texto = f"{tiempo} segundos" if tiempo is not None else "Sin récord"

        # Etiqueta para la dificultad
        etiqueta_dificultad = tk.Label(frame, text=f"{dificultad}:", font=("Arial", 14), bg="#f0f0f0", fg="#333333")
        etiqueta_dificultad.grid(row=i + 1, column=0, sticky="w", pady=5)  # Usar grid para las etiquetas

        # Etiqueta para el tiempo
        etiqueta_tiempo = tk.Label(frame, text=tiempo_texto, font=("Arial", 14), bg="#f0f0f0", fg="#333333")
        etiqueta_tiempo.grid(row=i + 1, column=1, sticky="w", pady=5)  # Usar grid para las etiquetas

    # Botón para cerrar la ventana
    boton_cerrar = tk.Button(frame, text="Cerrar", font=("Arial", 14), bg="#f44336", fg="white", command=ventana_records.destroy)
    boton_cerrar.grid(row=len(DIFICULTADES) + 1, column=0, columnspan=2, pady=20)  # Usar grid para el botón
```

- **volver_al_menu() :** Se utiliza después de completar una partida para limpiar la interfaz actual y volver al menú principal.

```python
def volver_al_menu(self):
    """Vuelve al menú principal."""
    for widget in self.root.winfo_children():
        widget.destroy()
    self.crear_menu_dificultad()
```

- **Constantes Globales** :
  - DIFICULTADES: Define las configuraciones iniciales (filas, columnas, número de minas) para cada nivel.
  - COLORES_FONDO: Asigna colores a las celdas según el número de minas adyacentes.
  - RECORDS_FILE: Nombre del archivo donde se guardan los récords.

```python
# Configuración inicial del juego
DIFICULTADES = {
    "Facil": {"filas": 6, "columnas": 6, "minas": 6},
    "Medio": {"filas": 12, "columnas": 12, "minas": 30},
    "Dificil": {"filas": 16, "columnas": 16, "minas": 50}
}

# Paleta de colores para el fondo de los números
COLORES_FONDO = {
    1: "#e7f3fe",  # Azul claro
    2: "#c8e6c9",  # Verde claro
    3: "#ffcdd2",  # Rojo claro
    4: "#e1bee7",  # Púrpura claro
    5: "#ffe0b2",  # Naranja claro
    6: "#b2ebf2",  # Cian claro
    7: "#d7ccc8",  # Gris claro
    8: "#f5f5f5"   # Blanco grisáceo
}

# Archivo para guardar los récords
RECORDS_FILE = "records.json"
```

Esta estructura modular permite separar la lógica del juego de la interfaz gráfica, mejorando la legibilidad y escalabilidad del código.

2. Lógica de juego (minas, adyacencias, victoria/derrota)

Colocación de Minas

- Las minas se distribuyen aleatoriamente en el tablero mediante la función **colocar_minas()**. Se asegura de que no se superen el número máximo de minas definido para la dificultad seleccionada.

Cálculo de Adyacencias

- La función **calcular_adyacencias()** itera sobre cada celda del tablero y utiliza **contar_minas_adyacentes()** para determinar cuántas minas hay en las ocho posiciones circundantes. Esto permite mostrar números en las celdas que indican la proximidad de las minas.

Revelar Celdas

- Cuando el jugador hace clic en una celda, el método **revelar_celda()** comprueba si contiene una mina. Si es así, el juego termina mostrando un mensaje de derrota. Si no hay minas, se revela el número de minas adyacentes o, si no hay ninguna, se revelan recursivamente las celdas vecinas hasta encontrar una con minas cercanas.

Mecánicas de las Banderas e Interrogaciones

El uso de banderas e interrogaciones son unas herramientas clave para marcar celdas sospechosas de contener minas o que tengamos dudas y queramos resolver más tarde. El método **colocar_bandera()** gestiona la funcionalidad de las banderas:

- Cuando el jugador hace clic derecho en una celda no revelada, puede colocar o quitar una bandera.
- Las banderas son limitadas y están disponibles en un número igual al total de minas en el tablero. Esto evita que el jugador marque celdas indiscriminadamente.
- Si una celda ya tiene una bandera colocada, hacer clic derecho nuevamente la quita, permitiendo corregir errores.
- La interfaz muestra en todo momento cuántas banderas han sido usadas y cuántas quedan disponibles, proporcionando retroalimentación constante al jugador.

El método **colocar_interrogacion()** gestiona la funcionalidad de las interrogaciones:

- Cuando el jugador hace doble clic derecho en una celda no revelada, puede colocar o quitar una interrogación.
- El número de interrogaciones son infinitos, ya que no hay necesidad de imponer un límite
- Si una celda ya tiene una interrogación colocada, hacer clic derecho nuevamente la quita, permitiendo corregir errores.
- La celda no se revelará si se encuentra una interrogación puesta. Se debe eliminar para poder revelarla.

Victoria/Derrota

- **Derrota** : Ocurre cuando el jugador revela una celda con una mina. En este caso, el método **destapar_tablero_perdido()** muestra todas las minas y finaliza el juego.
- **Victoria** : Se alcanza cuando todas las celdas sin minas han sido reveladas. El método **verificar_victoria()** verifica esta condición y guarda el tiempo del jugador como un récord si es el mejor para esa dificultad.

3. Interfaz gráfica y usabilidad (Tkinter)

La interfaz gráfica está desarrollada utilizando la biblioteca tkinter, que proporciona herramientas para crear ventanas y widgets interactivos.

Menú Inicial

- El menú principal incluye:
  - Un título descriptivo.
  - Un Combobox para seleccionar la dificultad.
  - Botones para iniciar el juego, ver los récords o salir.

Tablero del Juego

- El tablero se representa mediante una cuadrícula de botones (Button) que corresponden a las celdas del juego.
- Los botones responden a clics izquierdos (revelar celda) y derechos (colocar/quitar banderas o interrogaciones).
- Se incluyen etiquetas para mostrar el tiempo transcurrido y el número de banderas disponibles.

Estilo Visual

- Se ha utilizado una paleta de colores moderna para mejorar la experiencia del usuario:
  - Colores distintivos para los números que indican minas adyacentes.
  - Fondo oscuro para el tablero y texto claro para mejorar la legibilidad.
  - Botones estilizados con gradientes y sombreados.

Usabilidad

- El diseño es intuitivo y fácil de usar, incluso para jugadores principiantes.
- Los mensajes emergentes (messagebox) proporcionan retroalimentación clara sobre el estado del juego.

4. Funcionalidades adicionales

- **Récords :** El programa guarda automáticamente los mejores tiempos en un archivo JSON (records.json) y permite visualizarlos en una ventana separada.
- **Contador de Tiempo :** Muestra el tiempo transcurrido durante el juego y se detiene automáticamente al ganar o perder.
- **Banderas :** Permite al jugador marcar celdas sospechosas con banderas, limitando su uso al número total de minas.
- **Interrogaciones :**
- **Volver al Menú :** Después de completar una partida, el jugador puede volver al menú principal para comenzar una nueva partida o salir.

5. Documentación, comentarios y uso de asistentes de código basados en IA

Uso de Asistentes de Código Basados en IA

- Durante el desarrollo, **se utilizó asistentes de IA** para optimizar la estructura del código, sugerir mejoras en la usabilidad y generar documentación adicional.
- Estos asistentes **ayudaron a identificar posibles errores y mejorar** la eficiencia del programa.
- También **ofrecieron las bases ya hechas sobre el proyecto** donde, a partir de este punto, empezaría a implementar diferentes funcionalidades y personalización propia.
- **Prompts:**
    - **Base para el juego:** "Quiero que hagas el juego del buscaminas en python usando la librería de tkinter. Primero implementa un menú de selección de dificultad (fácil, medio, difícil) y, dependiendo de este, el número de minas debe variar. Una vez se selecciona la dificultad y pulsamos sobre el botón de "jugar", que encontraremos en el menú de inicio, pasaremos a otra ventana que mostrará el tablero del buscaminas. Calcula la adyacencia de las minas, alrededor de estas y las funciones para revelar una celda"
    - **Función de añadir bandera:** "Ahora vamos a trabajar en base a ese código, para que lo tengas en cuenta de aquí en adelante. Quiero que añadas la función del click derecho que permitirá colocar una bandera sobre una casilla tapada. También habrá un label en la pantalla del tablero donde indicará el número usado y número total de banderas que se irá actualizando conforme se coloque o quite las banderas. El número total de banderas será el de las minas que se encuentren en el tablero"
    - **Función para añadir sonidos:** "¿Cómo podría hacer que se reproduzca un efecto de sonido con Tkinter?. En mi caso, quiero que se reproduzca al desvelar una casilla que contenga una mina o se marque con una bandera o interrogación"

6. Bibliografía

He consultado diferentes fuentes de información, tanto manuales como videos, asistentes de IA, blogs, etc…

- **Algoritmo para rellenar:** https://www.youtube.com/watch?v=NU3MGDvNNnI
- **IA para asistente de código y funcionalidades:** Qwen.ai
- **Guía de Tkinter:** https://guia-tkinter.readthedocs.io/es/develop/chapters/5-basic/5.1-Intro.html
- **Video básico de Tkinter:** https://www.youtube.com/watch?v=MpkTYMzhV0A

7. Conclusiones y opiniones personales

Mi conclusión personal es que he alcanzado la idea de mi proyecto y la he mejorado un poco más, aunque no tanto como en realidad quería. He podido crear las bases del juego y añadir mis opciones extras como el menú de inicio y tabla de records.

Me ha servido mucho para aprender Tkinter por mi cuenta y trabajar con algo que nunca había visto e implementarlo en un juego. También me ha ayudado a entender el sistema de aplicaciones python por ventanas e interactivo para el usuario, ya que el programa tiene que ser sencillo para que cualquiera pueda usarlo. Además, he aprendido a implementar la lógica de python para llevarla a la parte de interfaz, lo cual me ha abierto mucho más mi manera de trabajar.

El desarrollo ha sido tedioso pero motivante por el simple hecho de tener que hacerlo por mi cuenta consultando toda fuente de información que pudiera e implementar todo lo que aprendiera en este proyecto, pero al ser mi primera vez trabajando de esta manera se me ha hecho más complicado.

8. Aspectos innovadores más allá de requisitos

Los aspectos que he podido implementar en mi programa son:

- **Tabla de records:** Esta almacena por cada dificultad el tiempo mínimo que ha tardado cualquier usuario en ganar. Esto trabaja con archivos JSON.
- **Menú por ventanas:** Menú interactivo de inicio que permite seleccionar la dificultad, jugar, salir o ver la tabla de records. Esto trabaja con el manejo de ventanas de Tkinter.
- **Sonidos interactivos:** Sonidos específicos para cada vez que el usuario desvela mina, bandera o interrogación.
- **Interrogación:** Función que permite marcar una casilla con una interrogación → ❔ en caso de duda de casilla para resolver más tarde. Esta se marca haciendo doble click derecho sobre una casilla.
- **Modularidad** : La estructura orientada a objetos facilita la modificación y expansión del juego con nuevas características.

Un par de aspectos que quise implementar y no pude son:

- **Records guardando nombre de usuario:** Sería la misma tabla de records pero el usuario debe introducir su nombre antes de empezar a jugar para que se quede registrado junto a su tiempo récord en la dificultad que haya escogido.
- **Histórico de partidas:** Existiría un historial de todas las partidas que se hayan jugado, pudiendo acceder a estas desde otra ventana seleccionándolas por fecha y hora, mostrando así el tablero final de esa partida ya se haya ganado o perdido.
- **Login Administrador o Jugador:** Por último me habría gustado implementar un sistema de login que permite al administrador entrar en un panel de administración para gestionar las diferentes características del programa, como niveles de dificultad o gestión de partidas. Para el usuario común solo podría acceder al apartado de jugar.