PROYECTO FINAL

Equipo:
Cerón Samperio Lizeth Montserrat
Higuera Pineda Angel Abraham
Lorenzo Silva Abad Rey
Moya Rivera Mía Paulina

Indice

Objetivo del proyecto

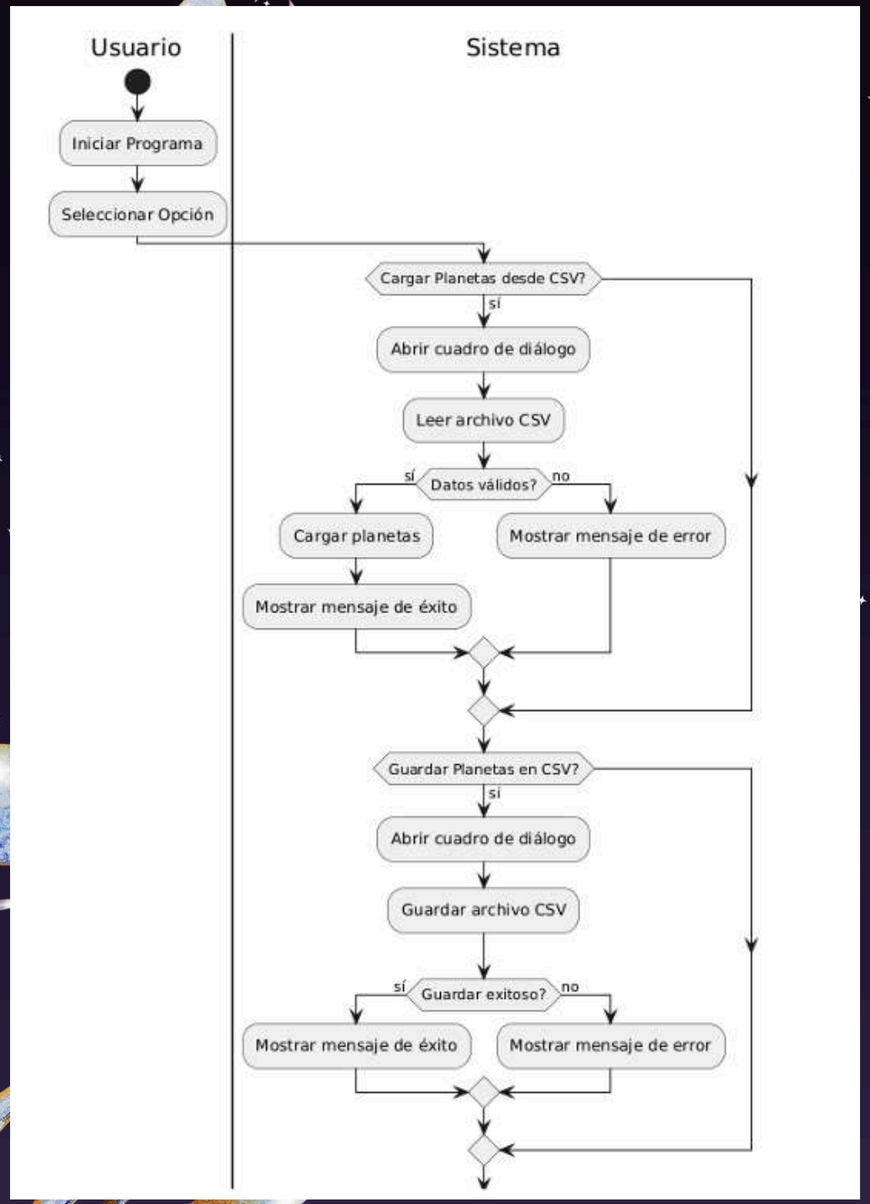
Desarrollar un programa que simule el movimiento orbital de planetas en un sistema solar utilizando matrices de rotación y escalado, para calcular y visualizar trayectorias elípticas en tiempo real.

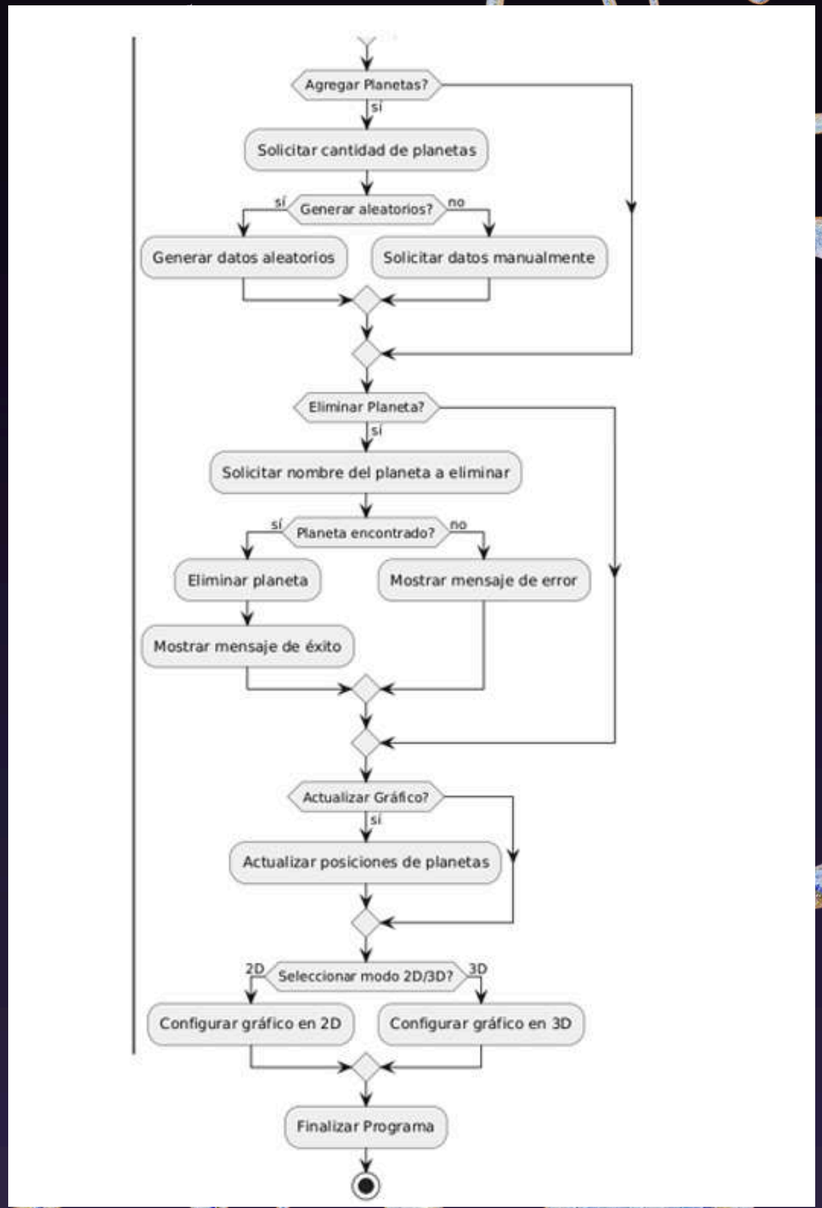
Descripción

El programa debe cumplir con las siguientes funciones:

- Representar un sistema solar con un sol fijo y varios planetas orbitando a su alrededor.
- Utilizar matrices de rotación para el movimiento orbital de planetas y matrices de escalado para ajustar el tamaño de las órbitas.
- Visualizar trayectorias orbitales de los planetas en un gráfico interactivo.
- Permitir al usuario personalizar los parámetros de los planetas como el radio orbital, velocidad angular y excentricidad.
- Generar gráficos que muestren el comportamiento de las órbitas, con opción de visualizar varios planetas al mismo tiempo.

Diagrama de flujo





Desarrollo

```
5   import numpy as np
6   import matplotlib.pyplot as plt
7   import matplotlib.animation as animation
8   from mpl_toolkits.mplot3d import Axes3D
9   import tkinter as tk
10  from tkinter import messagebox, filedialog
11  import csv
12  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
13  import os
14  import random
15  import subprocess
```

Primero, se importan las bibliotecas para manipular datos “numpy” y “csv”, para visualización “matplotlib” y para la creación de interfaces gráficas. “tkinter”.

```
18  planetas = []
19  puntos = []
20  trayectorias = []
```

Aquí se definen listas para almacenar información sobre los planetas y sus trayectorias.

```python
def cargar_planetas_desde_csv():
    global planetas
    # Abrir un cuadro de dialogo para seleccionar el archivo
    file_path = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
    if file_path:
        try:
            with open(file_path, newline='') as csvfile:
                reader = csv.DictReader(csvfile)
                planetas.clear()    # Limpiar la lista de planetas antes de cargar nuevos datos
                for row in reader:
                    try:
                        # Extraer datos de cada fila del archivo CSV
                        nombre = row['Planeta']
                        radio = float(row['Radio'])
                        velocidad_angular = float(row['Velocidad Angular'])
                        escala = float(row['Escala'])
                        # Agregar los datos a la lista de planetas
                        planetas.append({"Planeta": nombre, "Radio": radio, "Velocidad Angular": velocidad_angular, "Escala": escala})
                    except ValueError:
                        # En caso de que ocurra un eror, mostara este mensaje
                        messagebox.showerror("Error", f"Datos invalidos en el archivo CSV: {row}")
                        continue
                messagebox.showinfo("Exito", "Planetas cargados exitosamente desde el archivo CSV.")
                regenerar_planetas()    # Actualizar la visualizacion con los nuevos datos
        except Exception as e:
            # En caso de que ocurra un eror, mostara este mensaje
            messagebox.showerror("Error", f"No se pudo leer el archivo CSV: {e}")
```

Esta función permite al usuario seleccionar un archivo CSV que contenga los datos sobre los planetas: nombre, radio, velocidad angular y escala. Estos datos se cargan en la lista planetas. Si hay un error en el formato del archivo, es decir, los datos no son numéricos, entonces se muestra el mensaje de error.

```python
def guardar_planetas_en_csv():
    # Abre un cuadro de dialogo para seleccionar la ubicación y el nombre del archivo
    file_path = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CSV files", "*.csv")])
    if file_path:
        try:
            with open(file_path, mode='w', newline='') as csvfile:
                fieldnames = ['Planeta', 'Radio', 'Velocidad Angular', 'Escala']
                writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
                writer.writeheader()
                for planeta in planetas:
                    writer.writerow(planeta)    # Guarda los datos de cada planeta y marcara en un cuadro de exito si se guardaron los datos
                messagebox.showinfo("Exito", "Planetas guardados exitosamente en el archivo CSV.")
        except Exception as e:
            # En caso de que ocurra un eror, mostara este mensaje
            messagebox.showerror("Error", f"No se pudo guardar el archivo CSV: {e}")
```

Esta función permite al usuario guardar la lista actual de planetas en un archivo CSV. También se asegura de manejar excepciones para asegurar que se notifique al usuario si ocurre un error durante el guardado.

```python
def configurar_grafico():
    global ax, info_frame
    # Limpiar la figura actual para eliminar basura, en caso de que tenga
    fig.clear()
    fig.patch.set_facecolor("#2f2f2f")  # Establecer el color de fondo a negro
    # Configuración para modo 3D
    if modo_3d.get():
        ax = fig.add_subplot(111, projection="3d")
        ax.set_xlim([-200, 200])
        ax.set_ylim([-200, 200])
        ax.set_zlim([-200, 200])
        ax.set_title("Simulación 3D de Órbitas Planetarias", color="white")
        if 'info_frame' in globals():
            info_frame.place_forget()  # Ocultar el cuadro de información en modo 3D para no tener errores
    else:
        # Configuración para modo 2D
        ax = fig.add_subplot(111)
        ax.set_xlim([-200, 200])
        ax.set_ylim([-200, 200])
        ax.set_title("Simulación 2D de Órbitas Planetarias", color="white")

    ax.set_facecolor("#2f2f2f") # Color de fondo del gráfico a negro
```

```python
    if not modo_3d.get():
        for spine in ax.spines.values():
            spine.set_color("white")    # Color de las etiquetas de los ejes
    # Mantiene la mayoria de parametros en color blanco
    ax.tick_params(colors="white")
    ax.yaxis.label.set_color("white")
    ax.xaxis.label.set_color("white")
    if modo_3d.get():
        ax.zaxis.label.set_color("white")
    # Mantiene en color gris las orbitas
    ax.grid(color="gray", linestyle="--")
    # Genera un circulo de color amarillo llamado sol
    ax.scatter(0, 0, color="yellow", s=200, label="Sol")

    canvas.draw()
```

Configura el gráfico según si se está utilizando el modo 2D o 3D. Visualmente establece límites y títulos del gráfico, cambia el color de fondo y otros elementos visuales para mejorar la presentación.

```python
def rotacion_3d(punto, angulo, eje='z'):
    """Aplica rotación en 3D a un punto."""
    x, y, z = punto
    angulo_rad = np.radians(angulo)
    if eje == 'z':
        matriz_rotacion = np.array([
            [np.cos(angulo_rad), -np.sin(angulo_rad), 0],
            [np.sin(angulo_rad), np.cos(angulo_rad), 0],
            [0, 0, 1]
        ])
    # Rotación en otros ejes se puede agregar aquí (X o Y).
    nuevo_punto = matriz_rotacion @ np.array([x, y, z])
    return nuevo_punto

def rotacion_2d(punto, angulo):
    """Aplica rotación en 2D a un punto."""
    x, y = punto
    angulo_rad = np.radians(angulo)
    matriz_rotacion = np.array([
        [np.cos(angulo_rad), -np.sin(angulo_rad)],
        [np.sin(angulo_rad), np.cos(angulo_rad)]
    ])
    nuevo_punto = matriz_rotacion @ np.array([x, y])
    return nuevo_punto
```

```python
def escalado_2d(punto, escala_x, escala_y):
    """Aplica escalado en 2D a un punto."""
    x, y = punto
    matriz_escalado = np.array([
        [escala_x, 0],
        [0, escala_y]
    ])
    nuevo_punto = matriz_escalado @ np.array([x, y])
    return nuevo_punto

def escalado_3d(punto, escala_x, escala_y, escala_z):
    """Aplica escalado en 3D a un punto."""
    x, y, z = punto
    matriz_escalado = np.array([
        [escala_x, 0, 0],
        [0, escala_y, 0],
        [0, 0, escala_z]
    ])
    nuevo_punto = matriz_escalado @ np.array([x, y, z])
    return nuevo_punto
```

En estas cuatro funciones es donde se implementan las matrices de transformación para la rotación y la escala en 2D y la función adicional de visualizar en 3D el sistema.

```python
def regenerar_planetas():
    global puntos, trayectorias, planetas, ax
    # Limpiar el eje actual
    ax.cla()
    # Configurar el gráfico nuevamente
    configurar_grafico()
    puntos.clear()
    trayectorias.clear()
    # Generar los puntos de la órbita circular
    for planeta in planetas:
        radio = planeta["Radio"]
        escala = planeta["Escala"]
        angulos = np.linspace(0, 2 * np.pi, 360)
        x_orbita = radio * np.cos(angulos) * escala
        y_orbita = radio * np.sin(angulos) * escala

        if modo_3d.get():
            z_orbita = np.zeros_like(x_orbita)  # En 3D, las órbitas están en el plano XY
            trayectorias.append(ax.plot(x_orbita, y_orbita, z_orbita, linestyle="--", color="white")[0])
            puntos.append(ax.scatter([], [], [], label=planeta["Planeta"]))
        else:
            trayectorias.append(ax.plot(x_orbita, y_orbita, linestyle="--", color="white")[0])
            puntos.append(ax.plot([], [], "o", label=planeta["Planeta"])[0])

    ax.legend()  # Mostrar leyenda con los nombres de los planetas
    canvas.draw()  # Redibujar el lienzo
```

Limpia el gráfico actual y vuelve a dibujar las órbitas de todos los planetas basándose en sus propiedades. Para el calculo de la órbita utiliza funciones trigonométricas para calcular las posiciones x e y de cada planeta en función de su radio y escala.

```python
def actualizar(frame):
    global puntos, planetas
    if actualizar_event.get():
        regenerar_planetas()
        actualizar_event.set(False)

    for i, planeta in enumerate(planetas):
        if i >= len(puntos):
            continue
        # Calcular la posición actual del planeta basado en su velocidad angular y el cuadro actual
        angulo_actual = (planeta["Velocidad Angular"] * frame) % 360
        x = planeta["Radio"] * np.cos(np.radians(angulo_actual)) * planeta["Escala"]
        y = planeta["Radio"] * np.sin(np.radians(angulo_actual)) * planeta["Escala"]
        z = 0

        if modo_3d.get():
            puntos[i]._offsets3d = ([x], [y], [z])  # Actualizar en 3D
        else:
            puntos[i].set_data([x], [y])            # Actualizar en 2D
    canvas.draw()
    return puntos
```

Esta función actualiza las posiciones de los planetgas en cada cuadro de animación. Calcula la nueva posición utilizando la velocidad angular y actualiza los objetos gráficos correspondientes.

```python
def ajustar_limites(valor=None):
    global ax, slider_zoom
    zoom = slider_zoom.get()
    ax.set_xlim([-10 * zoom, 10 * zoom])
    ax.set_ylim([-10 * zoom, 10 * zoom])
    if modo_3d.get():
        ax.set_zlim([-10 * zoom, 10 * zoom])
    fig.canvas.draw_idle()
```

Ajusta los límites de visualización del gráfico según el zoom.

```python
def agregar_planetas():
    global agregar_mas_planetas, contador_planetas

    def solicitar_cantidad():
        nonlocal ventana_solicitar

        try:
            cantidad = int(entry_cantidad.get())
            aleatorios = var_aleatorios.get()  # Obtener el estado de la casilla de aleatorios
            if cantidad <= 0:
                raise ValueError
            ventana_solicitar.destroy()
            mostrar_ventana_agregar(cantidad, aleatorios)
        except ValueError:
            messagebox.showerror("Error", "Por favor, introduce un número válido.")
```

Permite al usuario agregar nuevos planetas manualmente o generarlos aleatoriamente.

```python
    ventana_solicitar = tk.Toplevel(root)
    ventana_solicitar.title("Cantidad de Planetas")
    ventana_solicitar.geometry("300x200") #Resolucion de la ventana
    # Infomracion que se le solicita al usuario
    tk.Label(ventana_solicitar, text="¿Cuantos planetas deseas agregar?", font=("Unbounded ExtraBold", 9)).pack(pady=10)
    entry_cantidad = tk.Entry(ventana_solicitar, font=("Arial", 12))
    entry_cantidad.pack(pady=5)

    # Casilla para elegir si los datos serán aleatorios
    var_aleatorios = tk.BooleanVar(value=False)
    tk.Checkbutton(
        ventana_solicitar,
        text="¿Generar datos aleatorios?",
        variable=var_aleatorios,
        font=("Unbounded ExtraBold", 9)
    ).pack(pady=10)
    #Boton de aceptar
    tk.Button(ventana_solicitar, text="Aceptar", command=solicitar_cantidad, font=("Martian Mono Condensed ExtraBold", 10), bg="#4f4f4f", fg="white").pack(pady=10

    ventana_solicitar.mainloop()
```

Aquí es donde se crea la ventana para solicitar al usuario la cantidad de planetas y si se quiere generar con datos aleatorios.

```python
def mostrar_ventana_agregar(cantidad, aleatorios):
    global contador_planetas
    contador_planetas = 0

    if aleatorios:
        # Generar todos los datos aleatorios de una vez
        for _ in range(cantidad):
            nombre = f"Planeta_{contador_planetas + 1}"
            radio = round(random.uniform(10, 500), 2)
            velocidad = round(random.uniform(0.1, 30), 2)
            escala = round(random.uniform(1, 3), 2)
            planetas.append({"Planeta": nombre, "Radio": radio, "Velocidad Angular": velocidad, "Escala": escala})
            contador_planetas += 1
        regenerar_planetas()
        messagebox.showinfo("Éxito", f"Se han agregado {cantidad} planetas con datos aleatorios exitosamente.")
        return
```

Esta función muestra la ventana de agregar planetas, en caso de que la opción elegida sea “aleatorios” llena los campos con números aleatorios.

```
226     def guardar_datos():
227         try:
228             # Recuperar datos ingresados y agregarlos a la lista de planetas
229             nombre = entry_nombre.get()
230             radio = float(entry_radio.get())
231             velocidad = float(entry_velocidad.get())
232             escala = float(entry_escala.get())
233             planetas.append({"Planeta": nombre, "Radio": radio, "Velocidad Angular": velocidad, "Escala": escala})
234             regenerar_planetas()    # Actualizar el gráfico
235         except ValueError:
236             # En caso de tener un error, marca esto
237             messagebox.showerror("Error", "Datos inválidos. Por favor, revisa tus entradas.")
238     # Pasa al siguiente planeta en caso de que haya más de un plnaeta indicado para agregar
239     def siguiente_planeta():
240         nonlocal ventana_agregar
241         guardar_datos()
242         ventana_agregar.destroy()
243         mostrar_ventana_agregar(cantidad - 1, aleatorios)
244 
245     def terminar_agregar():
246         guardar_datos()  #
247         ventana_agregar.destroy()
248         messagebox.showinfo("Éxito", "Todos los planetas se han agregado exitosamente.")
```

También guarda los datos que son dados por el usuario y con estos datos actualiza el gráfico. En caso de que haya más planetas por agregar, pasa al siguiente y también guarda los datos. Al finalizar muestra una leyenda que indica que se ha realizado exitosamente.

```python
def eliminar_planeta():
    global planetas, puntos, trayectorias
    # Confirma si se elimino o no
    def confirmar_eliminacion():
        nombre = entry_nombre.get()
        for planeta in planetas:
            if planeta["Planeta"].lower() == nombre.lower():
                planetas.remove(planeta)
                messagebox.showinfo("Exito", f"Planeta '{nombre}' eliminado exitosamente.")
                regenerar_planetas()
                canvas.draw()  # Forzar redibujado del grafico para evitar errores de actualizacion
                ventana_eliminar.destroy()
                return
        messagebox.showerror("Error", f"No se encontro un planeta con el nombre '{nombre}'.")
    #Ventana de informacion
    ventana_eliminar = tk.Toplevel(root)
    ventana_eliminar.title("Eliminar Planeta")
    ventana_eliminar.geometry("300x150")

    tk.Label(ventana_eliminar, text="Nombre del Planeta a Eliminar:", font=("Unbounded ExtraBold", 10)).pack(pady=10)
    entry_nombre = tk.Entry(ventana_eliminar, font=("Arial", 12))
    entry_nombre.pack(pady=10)

    tk.Button(ventana_eliminar, text="Eliminar", command=confirmar_eliminacion, font=("Martian Mono Condensed ExtraBold", 9), bg="#4f4f4f", fg="white").pack(pady=10)
```

Esta función sirve para eliminar un planeta por su nombre, al dar el nombre del planeta a eliminar, da un mensaje que el programa lo ha hecho exitosamente y también borra la información de la ventanita donde aparece el nombre del planeta y su color.

Esta función detecta el mouse si apunta a la orbita de un planeta y al hacerlo, muestra la información del planeta al que se está apuntando.

```python
def on_hover(event):
    global panel_derecho, info_frame, info_label, planetas
    if not modo_3d.get() and event.inaxes == ax:
        x_event, y_event = event.xdata, event.ydata
        min_distance = float('inf')
        closest_planet = None

        # Verificar cercanía del cursor a las órbitas
        for planeta in planetas:
            radio = planeta["Radio"]
            escala = planeta["Escala"]

            # Calcular distancia a la órbita como distancia al círculo
            distancia_a_orbita = abs(np.sqrt(x_event**2 + y_event**2) - radio * escala)

            if distancia_a_orbita < 5:  # Umbral de cercanía a la órbita
                if distancia_a_orbita < min_distance:
                    min_distance = distancia_a_orbita
                    closest_planet = planeta

        # Mostrar información del planeta si está cerca de la órbita
        if closest_planet:
            if 'info_frame' not in globals():
                info_frame = tk.Frame(panel_derecho, bg="#ffffff", relief="solid", bd=1)
                info_label = tk.Label(info_frame, text="", font=("Arial", 12), bg="#ffffff", justify="left")
                info_label.pack(padx=5, pady=5)
            info_frame.place(relx=0.75, rely=0.85, anchor="center")
            info_label.config(text=f"Nombre: {closest_planet['Planeta']}\n"
                                   f"Velocidad: {closest_planet['Velocidad Angular']}\n"
                                   f"Radio: {closest_planet['Radio']}")
```

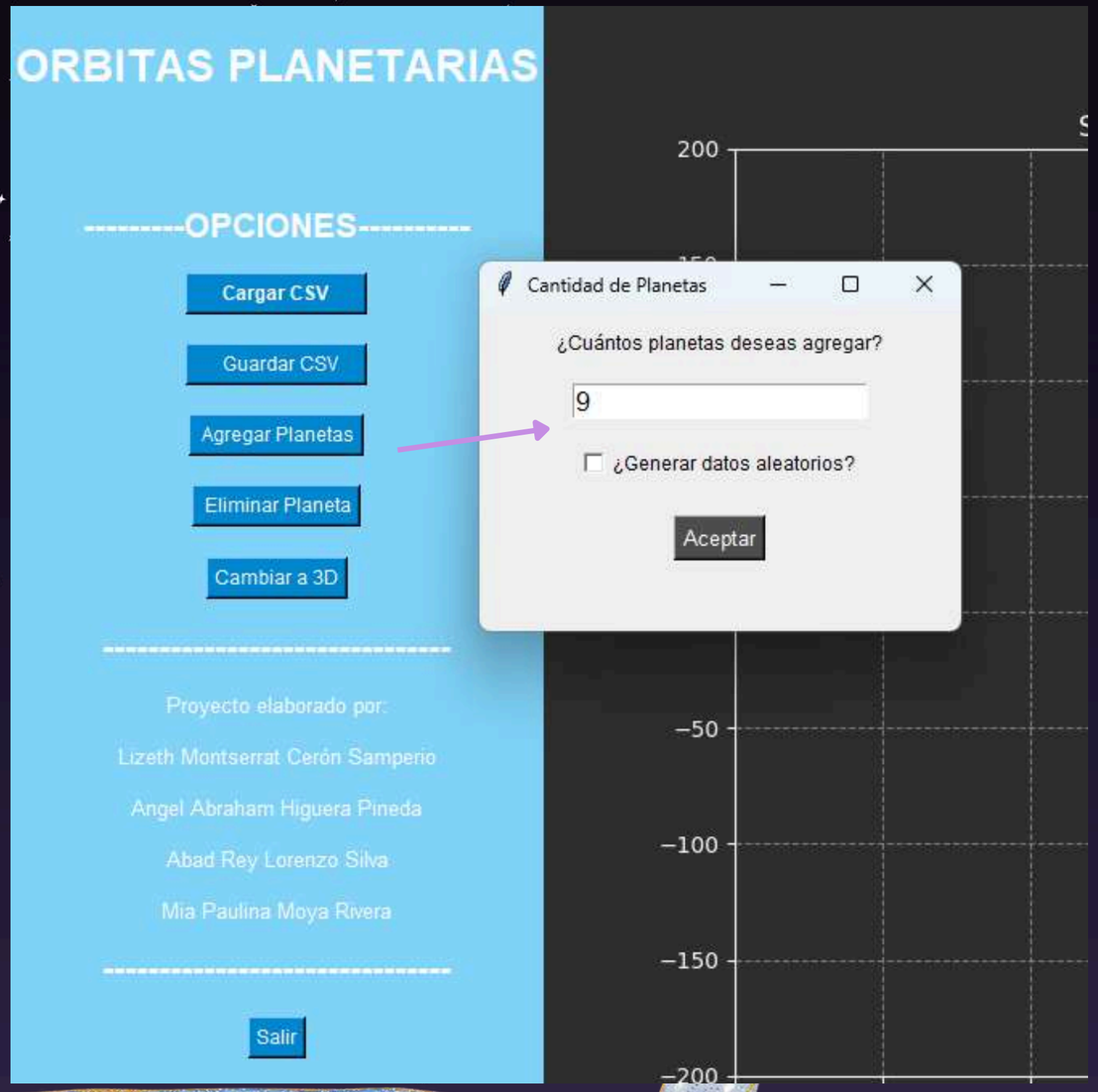
```python
def salir_con_video():
    """Muestra un video antes de salir del programa."""
    root.destroy()  # Cierra la ventana principal
```

Esta función cierra la ventana principal (anteriormente se tenia pensado llamar a un video, pero pór temas de optimización se cancelo)

```python
def iniciar_interfaz():
    global planetas, puntos, trayectorias, ax, fig, canvas, actualizar_event, modo_3d, slider_zoom, root, panel_derecho, info_frame, info_label

    root = tk.Tk()
    root.title("Simulación de Órbitas Planetarias") # Titulo
    root.geometry("1500x900")                       # resolucion
    root.configure(bg="#e0f7fa")                    # Color


    actualizar_event = tk.BooleanVar(value=False)
    modo_3d = tk.BooleanVar(value=False)

    # Panel izquierdo con sus botones y texto
    panel_izquierdo = tk.Frame(root, bg="#81d4fa", width=200)
    panel_izquierdo.pack(side="left", fill="y")
    #Texto
    tk.Label(panel_izquierdo, text="ORBITAS PLANETARIAS", font=("Unbounded ExtraBold", 20, "bold"), bg="#81d4fa", fg="white").pack(pady=20)
    tk.Label(panel_izquierdo, text=" ", font=("Arial", 16, "bold"), bg="#81d4fa", fg="white").pack(pady=5)
    tk.Label(panel_izquierdo, text="---------OPCIONES----------", font=("Unbounded ExtraBold", 16, "bold"), bg="#81d4fa", fg="white").pack(pady=5)
    #Botones
    tk.Button(panel_izquierdo, text="      Cargar CSV      ", command=cargar_planetas_desde_csv, bg="#0288d1", fg="white", font=("EXCRATCH", 9, "bold")).pack(pady=9)
    tk.Button(panel_izquierdo, text="      Guardar CSV    ", command=guardar_planetas_en_csv, bg="#0288d1", fg="white", font=("EXCRATCH", 9)).pack(pady=9)
    tk.Button(panel_izquierdo, text=" Agregar Planetas", command=agregar_planetas, bg="#0288d1", fg="white", font=("EXCRATCH", 9)).pack(pady=9)
    tk.Button(panel_izquierdo, text=" Eliminar Planeta", command=eliminar_planeta, bg="#0288d1", fg="white", font=("EXCRATCH", 9)).pack(pady=9)
    def cambiar_modo():
        modo_3d.set(not modo_3d.get())
        configurar_grafico()
        regenerar_planetas()
        cambiar_modo_boton.config(
            text="Cambiar a 3D" if not modo_3d.get() else "Cambiar a 2D"
        )
```

Esta función inicializa la interfaz gráfica del programa con botones para cargar archivos CSV, agregar o eliminar planetas. Utiliza un diseño simple con botones etiquetados.

Entrada del Usuario



Al presionar la opción de Agregar Planetas, se abrirá un recuadro que solicita especificar cuántos planetas se quieren agregar. También da la opción adicional de generar el planeta nuevo con datos aleatorios.

En este caso escogeremos 9 planetas.

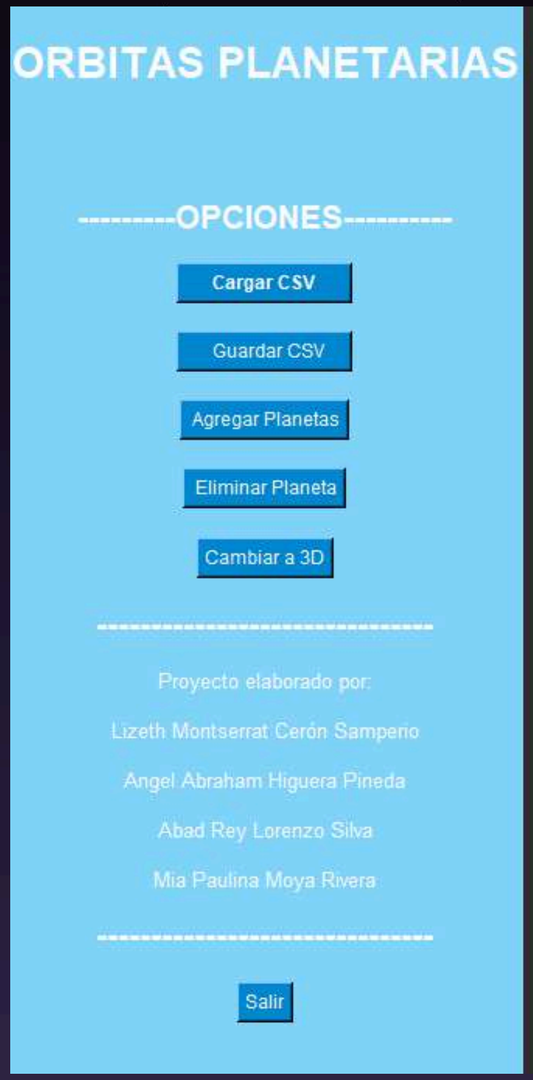
Al aceptar el recuadro anterior, se abre un nuevo recuadro donde podremos modificar los siguientes campos.

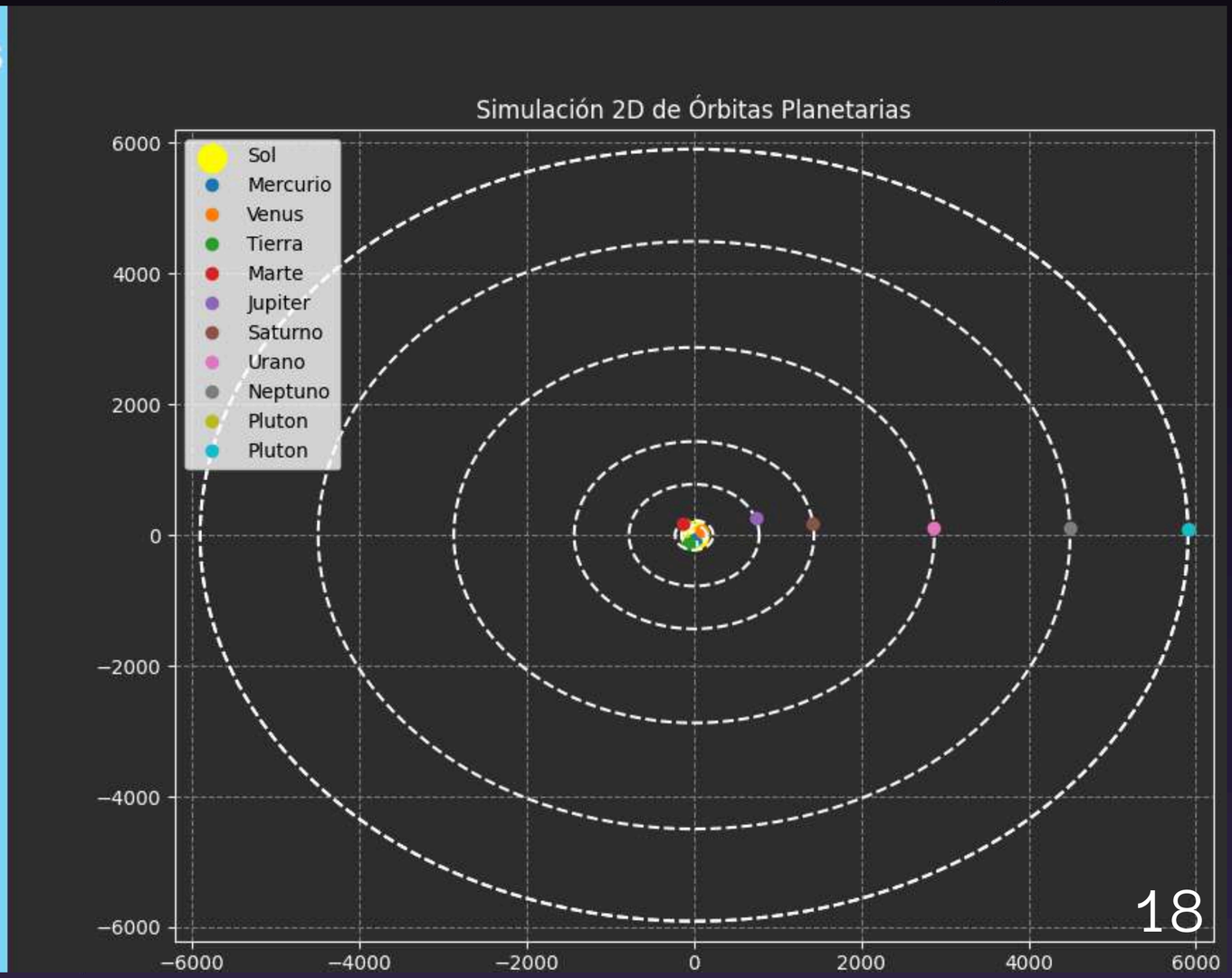
- Nombre del planeta.
- Radio orbital.
- Velocidad angular.
- Factor de escala.

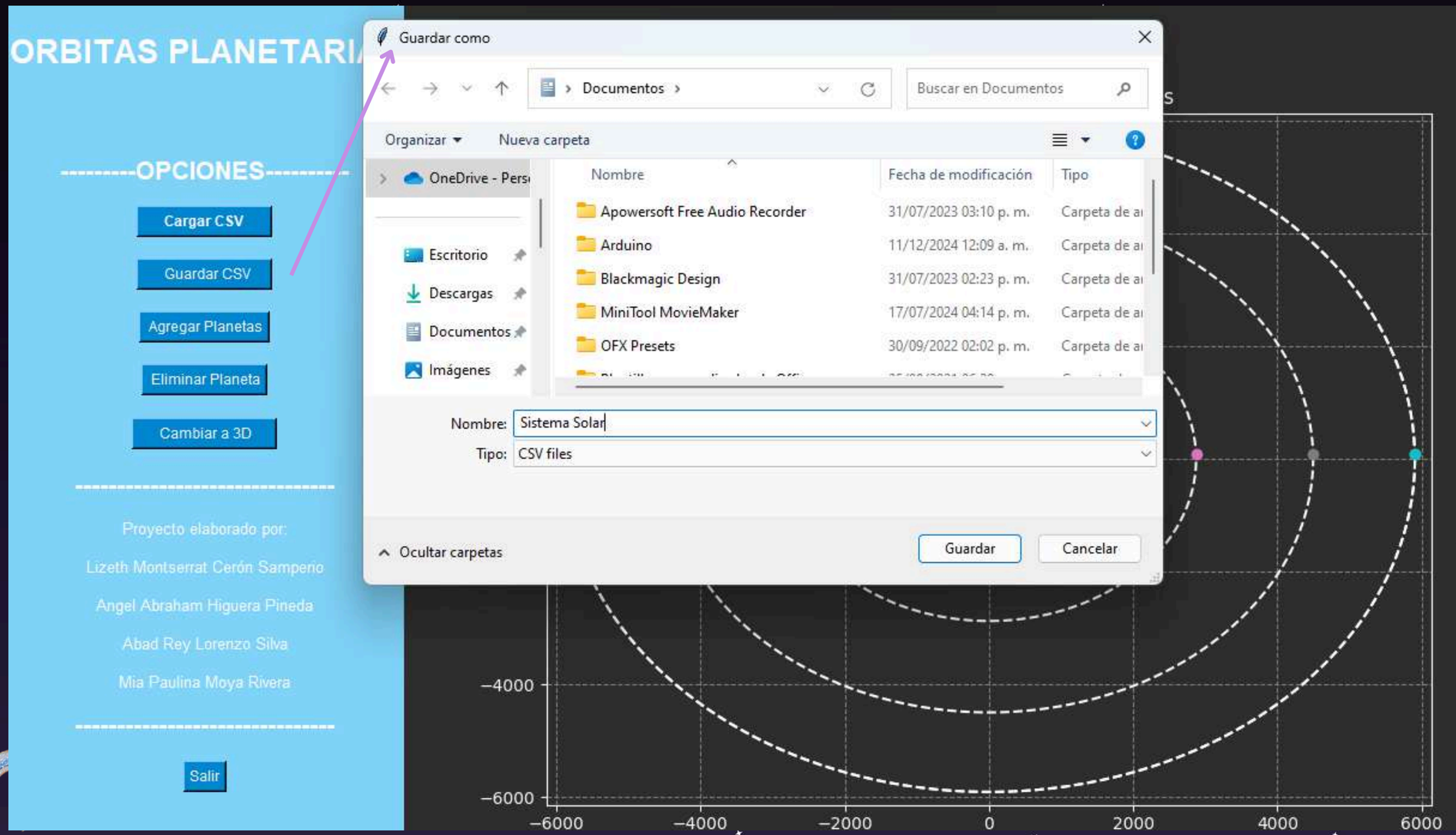
Podemos dar “Siguiente” hasta llenar los datos de los nueve planetas que especificamos anteriormente.

Resultado

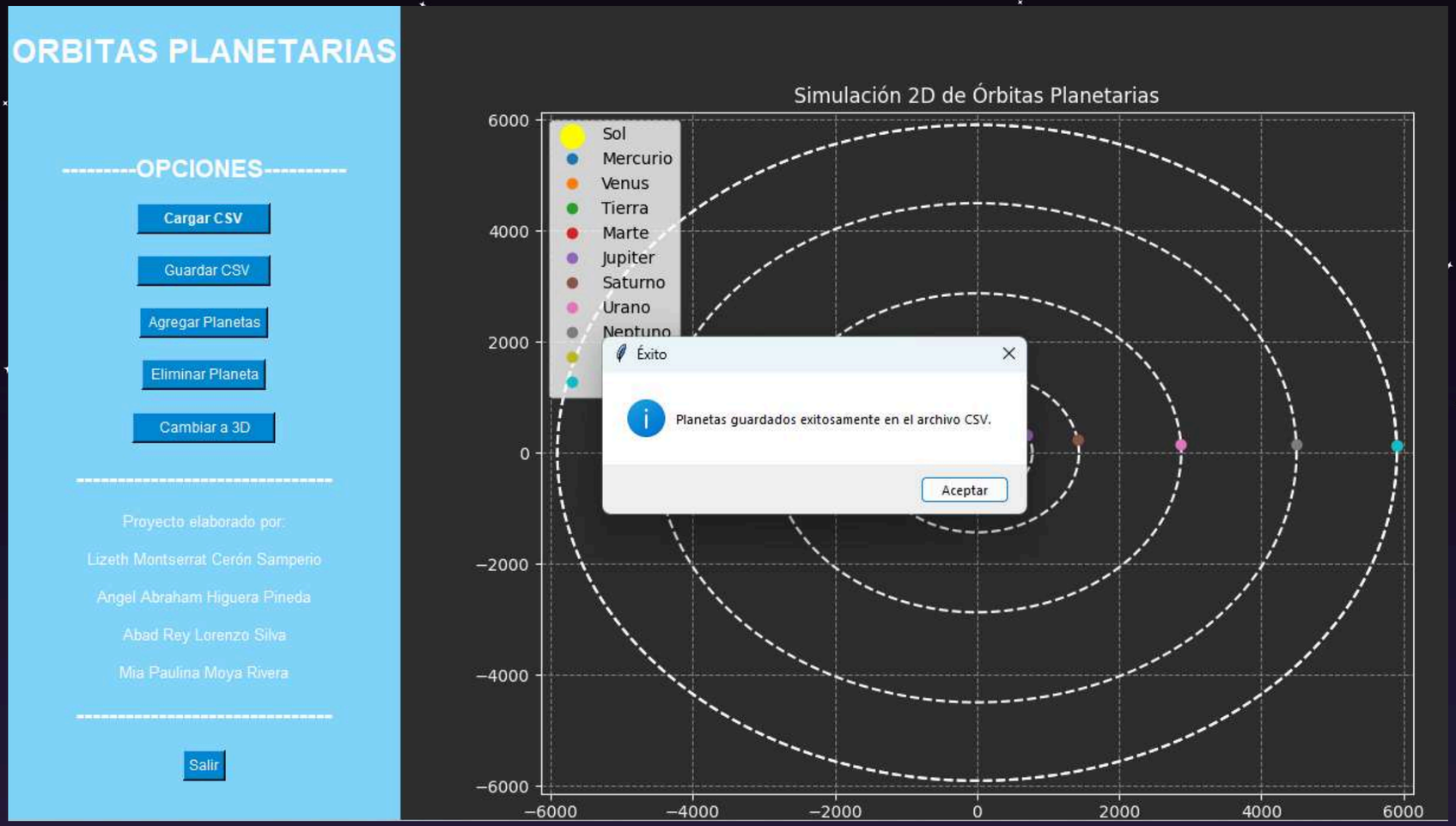
Podemos observar los planetas y su comparación con los otros planetas agregados. Algunos tienen orbitas más pequeñas y por eso no se alcanzan a ver.



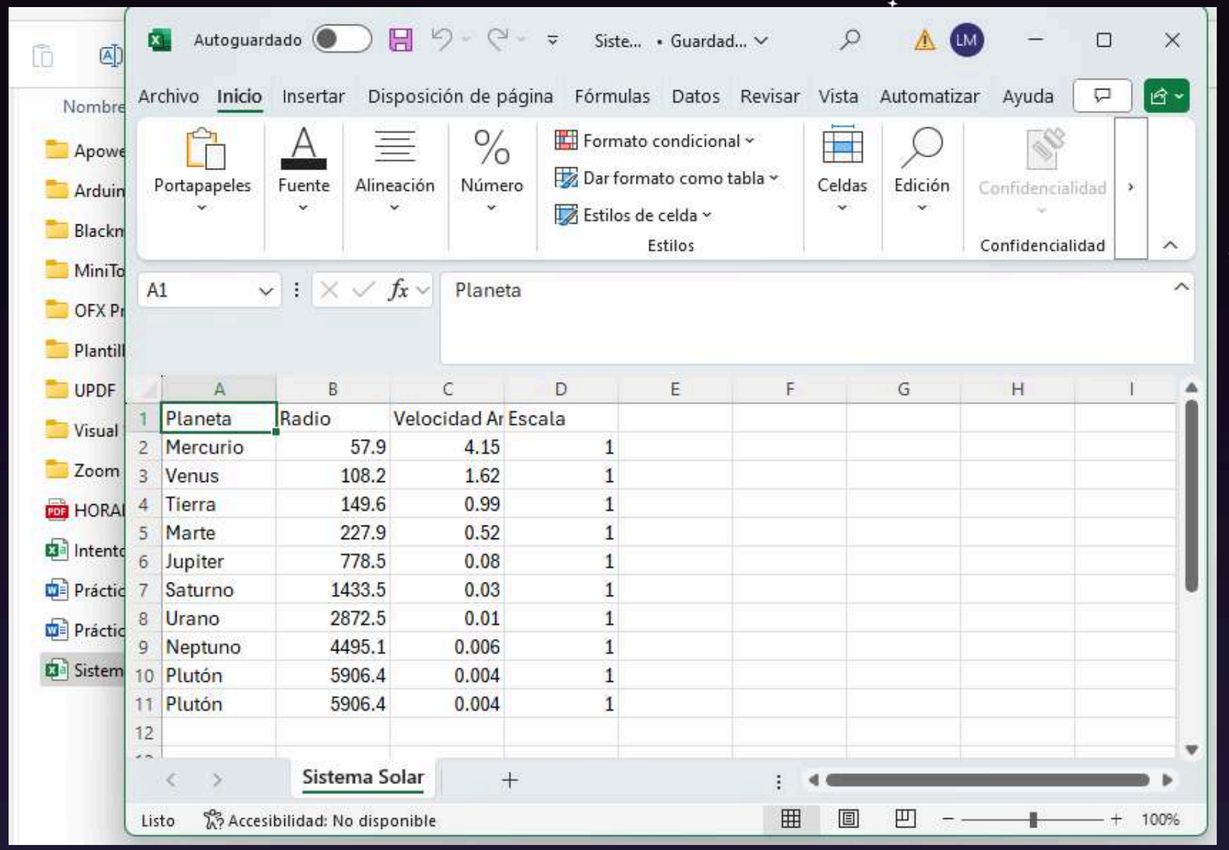




En la opción de Guardar CSV nos abrirá un recuadro para que le pongamos nombre al archivo y destinemos su ubicación.



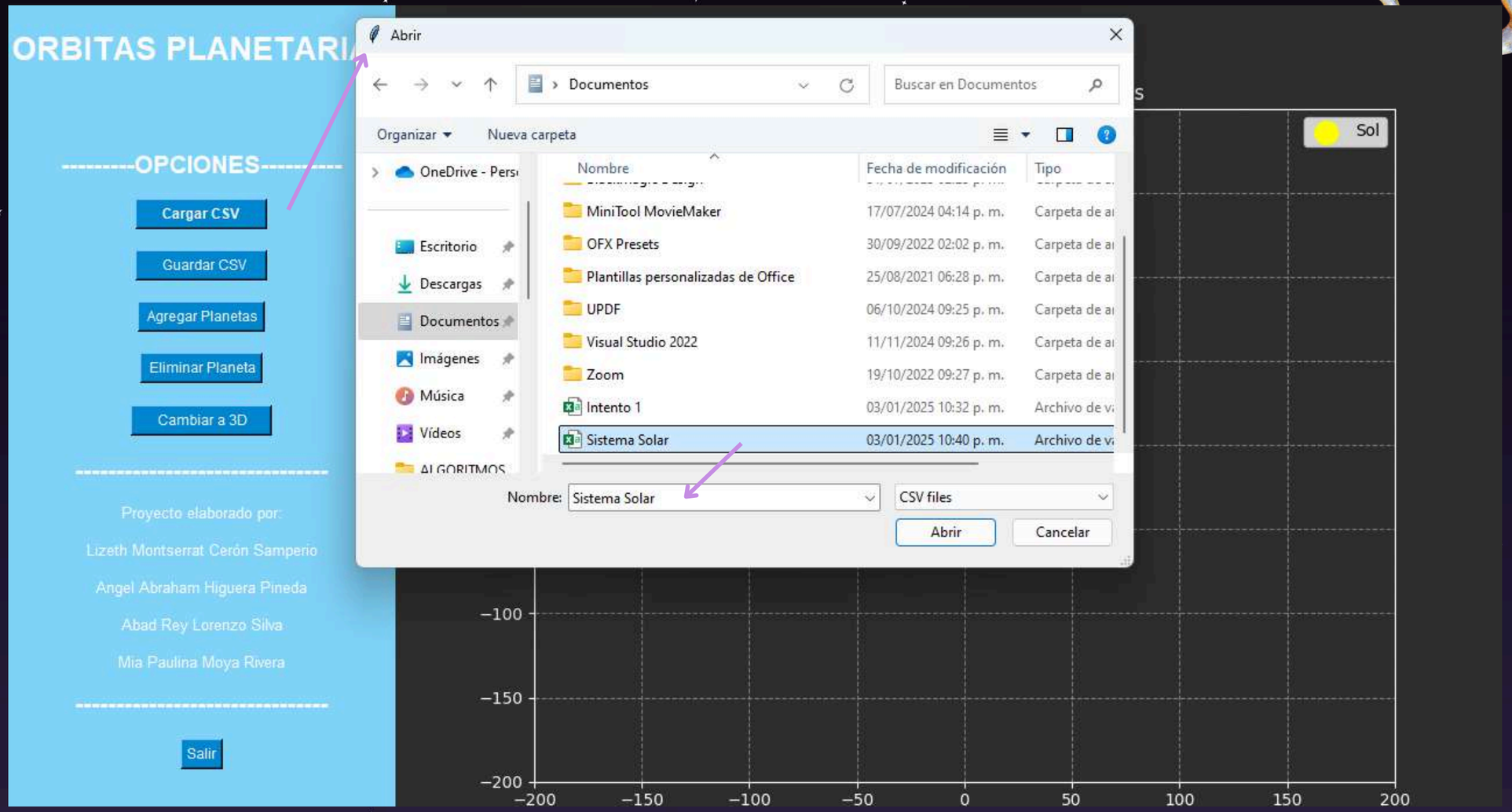
Una vez guardado el archivo CSV, nos aparecerá un mensaje que indica que la tarea se ha realizado exitosamente.

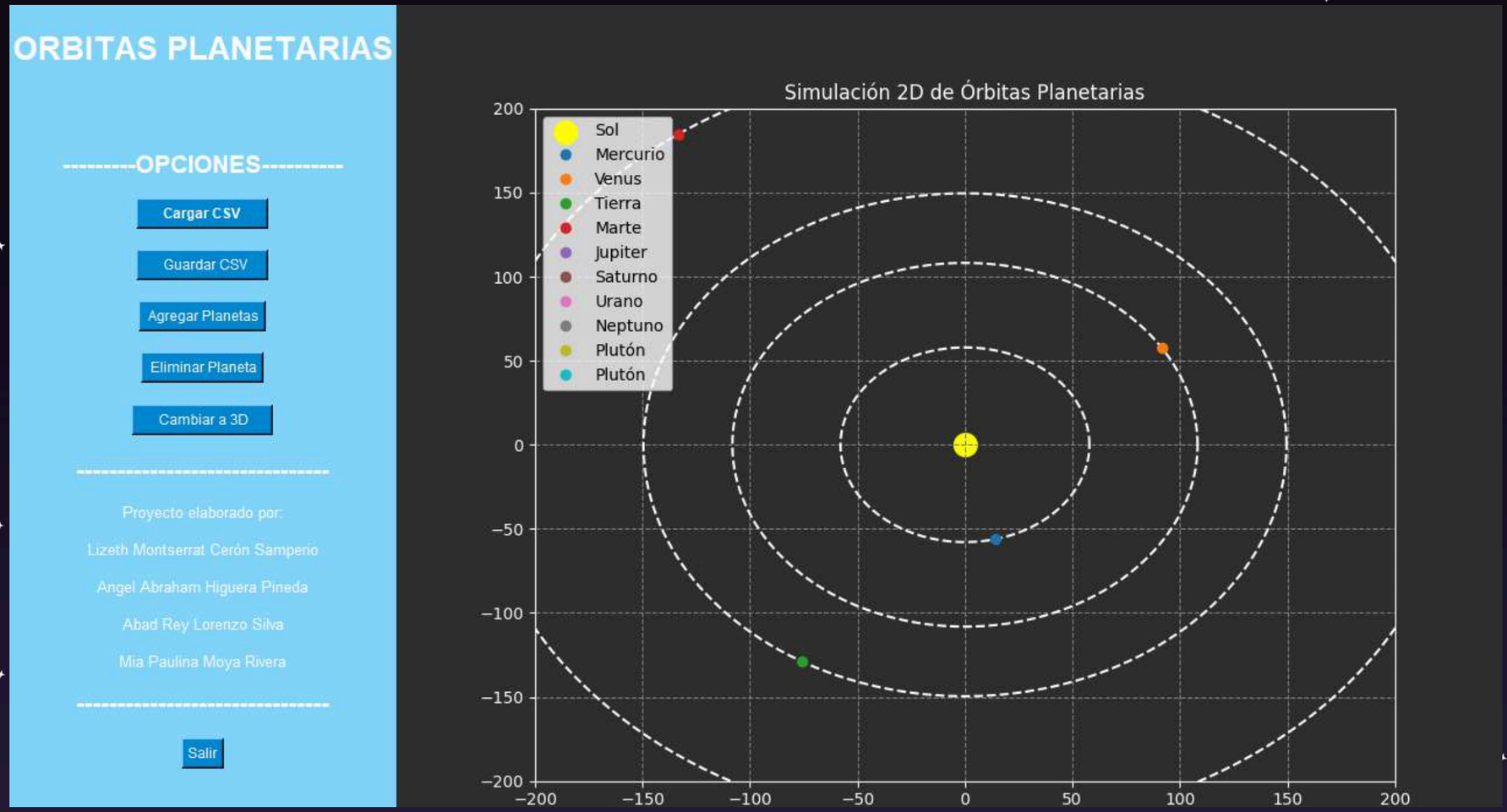


	A	B	C	D
1	Planeta	Radio	Velocidad Ar	Escala
2	Mercurio	57.9	4.15	1
3	Venus	108.2	1.62	1
4	Tierra	149.6	0.99	1
5	Marte	227.9	0.52	1
6	Jupiter	778.5	0.08	1
7	Saturno	1433.5	0.03	1
8	Urano	2872.5	0.01	1
9	Neptuno	4495.1	0.006	1
10	Plutón	5906.4	0.004	1
11	Plutón	5906.4	0.004	1
12				

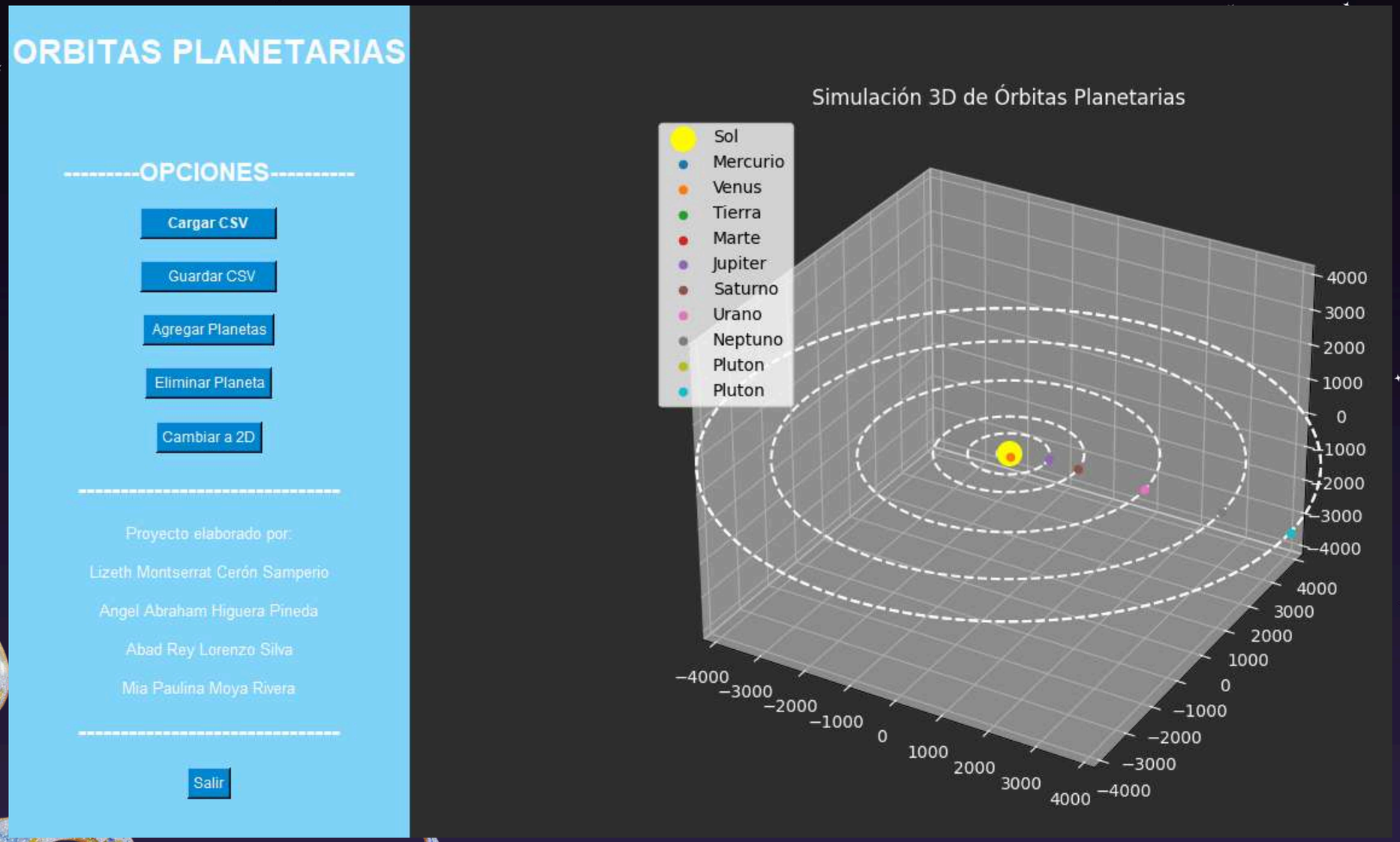
El archivo CSV se verá de la siguiente manera. Contendrá los datos de los planetas almacenados en el simulador en forma de tabla.

Para la opción de Cargar CSV, nos abrirá el explorador de archivos. En nuestro caso seleccionaremos el archivo de Sistema Solar, que contiene los 9 planetas y sus datos.





Una vez cargado el archivo, nuevamente visualizaremos los 9 planetas y sus órbitas.



Una opción adicional es la visualización del sistema solar en 3D.
El cuál se ve de esta manera:

Conclusiones

Se cumplen con los objetivos y los requisitos del proyecto ya que:

- Sí se visualizan las orbitas.
- Sí se muestra en consola los parámetros utilizados para cada planeta.
- Los gráficos sí permiten observar cómo varían las órbitas según los parámetros definidos.


GRACIAS