PROJECT REPORT

3D Solar System Explorer

Company Name
CodeX Computing Co.
Submitted to: Mudasir Naeem

Anas Ahmed Qureshi (CCPY-0006)
anasahmedcp@gmail.com

Table of Contents

1. Introduction 3

2. Project Objectives 3

3. Tools and Technologies Used 3

4. System Architecture & Design 4

   4.1 Architecture Overview 4

   4.2 GUI Design 4

   4.3 Key Features 4

5. Functional Description 5

   5.1 Real-Time Animation 5

   5.2 Planet Information Display 5

   5.3 Interactive Controls 5

   5.4 Data Loading & Persistence 5

6. Mathematical and Technical Concepts 6

7. Challenges and Solutions 6

8. Results 6

9. Learning Outcomes 7

10. Core Classes & Their Functions 7

   10.1 Planet Class 7

   10.2 Moon Class 8

11. Main Application: SolarSystemExplorer 9

   11.1 Initialization & Setup 9

      1. Window Setup: 9

      2. Pygame Mixer: 9

      3. Image Loading: 9

      4. Data Loading: 9

   11.2 GUI Components 9

   11.3 Core Functionality 10

      A. Animation System 10

      B. Rendering Engine 10

C. User Interaction 11
12. Comparison: Current vs. Improved Approach 11
13. Conclusion 11
14. References 12
15. Code Snippets 13
16. Output Snippets 21

1. Introduction

The **3D Solar System Explorer** is a Python-based educational simulation that visualizes our solar system in an interactive, animated 3D-like interface. The project is developed using **Tkinter**, Python's built-in GUI library, and enhanced with advanced mathematical modeling to simulate orbital dynamics and perspective-based 3D visualization. This application is designed primarily for students, educators, and astronomy enthusiasts to explore the structure, motion, and characteristics of planets and moons in an immersive way.

2. Project Objectives

- To develop a user-friendly, interactive simulation of the solar system.
- To represent planetary data using accurate scaled visuals and animation.
- To allow users to explore scientific facts about each celestial body.
- To promote STEM education through hands-on digital exploration.

3. Tools and Technologies Used

Tool / Library	**Purpose**
Python 3.x	Core programming language
Tkinter	GUI development (canvas, controls, menus)
Pillow (PIL)	Image processing for planets and rings
CSV Files	Store planetary and satellite data
Math & Random Libraries	Orbital mechanics and positioning
Threading	Smooth animation rendering
Datetime Module	Time simulation

Note: Although the initial proposal mentioned vPython, the project was implemented entirely using Tkinter with a custom 3D engine. This decision provided finer control over the animation, simplified the architecture, and avoided complex dependencies.

4. System Architecture & Design

4.1 Architecture Overview

- The application follows a **modular object-oriented design**, with three core classes:
    - Planet: Handles data and position of each planet.
    - Moon: Represents moons orbiting each planet.
    - SolarSystemExplorer: Main application controller (GUI, animation, logic).

4.2 GUI Design

- The main interface is a **Tkinter window** split into:
    - **Canvas Area** (Left): Displays the animated solar system.
    - **Control Panel** (Right): Includes buttons, sliders, and information text box.

4.3 Key Features

- **3D Simulation**: Achieved using 2.5D projection (X, Y, Z with perspective) on a 2D canvas.
- **Zoom & Pan**: Users can zoom in/out or drag the view to focus on regions.
- **Date Simulation**: Displays current simulated date/time during animation.
- **Interactive Info Panel**: Clicking on a planet opens a detailed info box.
- **Custom Themes**: Users can switch between dark, light, and blue themes.
- **Orbits, Moons, and Rings**: All are dynamically rendered using parametric calculations.

5. Functional Description

5.1 Real-Time Animation

- Each planet orbits the Sun at a different speed using angular motion logic.
- Orbits are elliptical or circular depending on inclination and eccentricity.
- A background thread runs the animation at 20 FPS to ensure smooth rendering.

5.2 Planet Information Display

- Clicking on a planet opens a textual panel showing:
  - Scientific data: Mass, diameter, orbital period, axial tilt, etc.
  - Moon count and names
  - Fun facts and trivia
  - Planet images (if available)

5.3 Interactive Controls

- **Play, Pause, Reset, Zoom** controls
- Keyboard shortcuts for quick access
- Toggle options for:
  - Orbit paths
  - Planet labels
  - Moons
  - 3D perspective

5.4 Data Loading & Persistence

- Reads planetary and moon data from CSV files.
- Allows saving/loading simulation states using JSON.
- Exports screenshot and save session data.

6. Mathematical and Technical Concepts

- **3D-like Projection**: Converts spherical coordinates into 2D with depth (z) affecting the vertical perspective.
- **Orbital Motion**: Angular movement using angle += speed * time_scale.
- **Axial Tilt and Inclination**: Used to visually rotate orbital planes.
- **Image Texturing**: Simulates lighting using pixel intensity shading.
- **Zoom and Pan Calculations**: Centering based on user interactions and scaling factors.

7. Challenges and Solutions

Challenge	Solution
vPython integration complexity	Replaced with custom 3D projection in Tkinter
Performance optimization	Used threading and redraw throttling
Lack of native 3D in Tkinter	Simulated depth using z-axis projections and angle-based rotation
Image scaling artifacts	Applied PIL-based resizing with high-quality filters

8. Results

- The application runs smoothly with interactive controls.
- All major features of the proposal have been fully implemented.
- Moons, rings, orbits, and 3D perspective are functioning accurately.
- Planet data is educational, engaging, and visually supported.
- UI is intuitive and flexible, suitable for both casual users and students.

9. Learning Outcomes

- Mastery of Python GUI development using Tkinter
- Practical understanding of orbital mechanics and projections
- Experience with real-time animation, threading, and simulation loops
- Enhanced problem-solving in adapting to tech constraints (e.g., replacing vPython)
- Integrated art and science through dynamic visualization and design

10. Core Classes & Their Functions

10.1 Planet Class

Purpose: Represents a celestial body (planet) with physical, orbital, and visual properties.

Attribute	Description	Why Used?	Alternative Approach	Possible Improvement
name	Planet name (e.g., "Earth")	Basic identification	Could use enum for planets	Add IAU naming conventions
radius	Scaled radius for visualization	Controls planet size in GUI	Dynamic scaling based on real size	Auto-scale based on window size
distance_from_sun	Scaled distance from sun	Controls orbit radius	Logarithmic scaling for realism	Adjustable scale factor
orbital_speed	Radians per frame for orbit	Determines movement speed	Real-time physics (Newtonian)	Use Kepler's laws for accuracy
color	Hex color for planet (fallback)	Visual representation	Use only textures, no color	Dynamic color based on temp
moons	Number of moons	Info display	List of `Moon` objects	Load from NASA API
facts	Fun facts about planet	Educational content	Link to Wikipedia	Add citations/sources

axial_tilt	Tilt in degrees (e.g., 23.5° for Earth)	3D rotation effect	Ignore tilt in 2D mode	Animate seasonal changes
orbital_inclination	Orbit tilt relative to ecliptic	Realistic 3D orbits	Assume 0° for simplicity	Adjustable view angles
has_rings	Boolean for ringed planets	Visual effect	Auto-detect from texture	Procedural ring generation
image_path	Path to planet texture	High-quality visuals	Use vector graphics	Support WebP format
mass, diameter, etc	Scientific properties	Detailed info panel	Load from CSV/API	Real-time data updates

Key Methods:

- **update_position():** Updates planet's location based on orbital mechanics.
- **reset_position():** Resets to initial state (for simulation reset).

10.2 Moon Class

Purpose: Represents a moon orbiting a planet.

Attribute	Description	Why Used?	Alternative Approach	Possible Improvement
name	Moon name (e.g., "Moon")	Identification	Use NASA IDs	Add discovery details
planet	Parent planet object	Orbital reference	Store planet name only	Support binary systems
radius	Scaled size	Visual size	Relative to planet	Real-scale rendering
distance	Distance from planet	Orbit radius	Dynamic tidal forces	Simulate Lagrange points
orbital_speed	Movement speed	Animation	Physics-based	Perturbation effects
color	Fallback color	If no texture	Grayscale based on albedo	Procedural generation
facts	Moon information	Education	Link to database	Add crater maps

Key Methods:

- **update_position():** Adjusts moon's position relative to its planet.
- **reset_position():** Returns to starting location.

11. Main Application: SolarSystemExplorer

11.1 Initialization & Setup

Key Steps:

1. Window Setup:

- 1400x900 resolution, dark theme (#121212).
- Why? Space-themed UI, better contrast for stars.
- Alternative: Responsive layout.

2. Pygame Mixer:

- Background music support.
- Why? Better than playsound for looping.
- Alternative: Use simpleaudio for lightweight option.

3. Image Loading:

- Sun, planets, rings loaded via PIL (ImageTk.PhotoImage).
- Why? Tkinter-compatible format.
- Alternative: Use OpenCV for advanced effects.

4. Data Loading:

- Planets from *planets.csv*, moons from *satellites.csv*.
- Why? Easy to modify without code changes.
- Alternative: SQLite database.

11.2 GUI Components

Component	Description	Why Used?	Alternative	Improvement

Canvas	1100x700 drawing area	Tkinter's built-in graphics	Pygame/QtCanvas	GPU acceleration
Control Panel	Right sidebar (250px)	Easy access to controls	Collapsible panel	Dark/light mode
Music Controls	Play/pause, volume	Interactive experience	Remove if not needed	Spotify API integration
Time Scale Slider	0.1x to 5.0x speed	Adjust simulation speed	Fixed increments	Logarithmic scaling
View Options	Toggles (orbits, names, 3D)	Customizable UI	Presets (e.g., "Education Mode")	Save preferences
Planet ComboBox	Dropdown for planet info	Easy selection	Click-to-select on canvas	Search functionality

11.3 Core Functionality

A. Animation System

- **Threaded Animation**
    - Runs in a separate thread (threading.Thread).
    - Why? Prevents GUI freezing.
    - Alternative: root.after() for simpler logic.

- **Time Scaling**
    - Adjusts planet.update_position(time_scale).
    - Improvement: Add acceleration/deceleration.

B. Rendering Engine

- **Starfield Background**
    - 100 random white dots (canvas.create_oval).
    - Why? Simple space effect.
    - Alternative: Use Milky Way panorama.

- **3D Effects**
    - Z-axis depth (y += z * depth_factor).
    - Why? Fake 3D without OpenGL.
    - Alternative: Use pygame for real 3D.

- **Planet Textures**
    - Loaded via Pillow → ImageTk.PhotoImage.
    - Why? Tkinter compatibility.
    - Alternative: Use OpenCV for dynamic lighting.

C. User Interaction

Feature	Implementation	Why?	Improvement
Click & Drag	canvas.bind("<B1-Motion>")	Pan the view	Inertia scrolling
Mouse Wheel Zoom	canvas.bind("<MouseWheel>")	Zoom in/out	Smooth zooming
Planet Selection	get_planet_at_position()	Click-to-info	Tooltip previews

12. Comparison: Current vs. Improved Approach

Feature	Current Implementation	Better Alternative	Reason
3D Rendering	Fake depth with z-axis	Pygame 3D`/`OpenGL	Realistic shadows
Physics	Fixed orbital speed	Newtonian gravity	More accurate
Data Loading	CSV files	SQLite/NASA API	Live updates
UI Framework	Tkinter	PyQt/PySide	Modern look
Textures	Static images	Procedural generation	Dynamic weather

13. Conclusion

The **3D Solar System Explorer** is a successful demonstration of using Python to create an interactive, educational, and visually rich software application. By blending scientific accuracy with user-focused design, the project fulfills its educational purpose while offering extensibility for future enhancements like sound, VR support, or deeper planetary exploration.

14. References

- NASA Planetary Fact Sheets: https://nssdc.gsfc.nasa.gov/planetary/factsheet/
- Tkinter Documentation: https://docs.python.org/3/library/tkinter.html
- Pillow (PIL): https://pillow.readthedocs.io/
- Python Math Module: https://docs.python.org/3/library/math.html
- Astronomy Articles and Wikipedia

15. Code Snippets

```python
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import math
import time
import threading
from PIL import Image, ImageTk, ImageOps
import os
import csv
from collections import defaultdict
import random
from datetime import datetime, timedelta
import io
import pygame  # Added for background music


1 usage
class Planet:
    """Class to represent a planet with its properties and orbital data"""

    def __init__(self, name, radius, distance_from_sun, orbital_speed, color,
                 moons, facts, axial_tilt, orbital_inclination, has_rings=False,
                 image_path=None, mass=None, diameter=None, density=None,
                 gravity=None, escape_velocity=None, rotation_period=None,
                 length_of_day=None, perihelion=None, aphelion=None,
                 orbital_period=None, orbital_velocity=None,
                 orbital_eccentricity=None, obliquity_to_orbit=None,
                 mean_temperature=None, surface_pressure=None,
                 has_ring_system=None, has_global_magnetic_field=None):
        self.name = name
        self.radius = radius  # Scaled radius for visualization
        self.distance_from_sun = distance_from_sun  # Scaled distance
        self.orbital_speed = orbital_speed  # Radians per frame
        self.color = color
        self.moons = moons
        self.facts = facts
        self.axial_tilt = axial_tilt  # Axial tilt in degrees
        self.orbital_inclination = orbital_inclination  # Orbital inclination in degrees
        self.has_rings = has_rings  # Whether the planet has rings

        # Position and orientation
```

```python
class Planet:
    def __init__(self, name, radius, distance_from_sun, orbital_speed, color,
        # Position and orientation
        self.angle = random.uniform(0, 2 * math.pi)  # Random starting position
        self.initial_angle = self.angle  # Store initial angle for reset
        self.x = 0
        self.y = 0
        self.z = 0  # For 3D depth
        self.rotation_angle = 0  # For planet rotation

        # Image handling
        self.image_path = image_path
        self.image = None  # Will store the processed image
        self.photo_image = None  # Will store the PhotoImage for tkinter
        self.texture_image = None  # For 3D texture mapping

        # Scientific properties
        self.mass = mass  # 10^24 kg
        self.diameter = diameter  # km
        self.density = density  # kg/m³
        self.gravity = gravity  # m/s²
        self.escape_velocity = escape_velocity  # km/s
        self.rotation_period = rotation_period  # hours
        self.length_of_day = length_of_day  # hours
        self.perihelion = perihelion  # 10^6 km
        self.aphelion = aphelion  # 10^6 km
        self.orbital_period = orbital_period  # days
        self.orbital_velocity = orbital_velocity  # km/s
        self.orbital_eccentricity = orbital_eccentricity
        self.obliquity_to_orbit = obliquity_to_orbit  # degrees
        self.mean_temperature = mean_temperature  # °C
        self.surface_pressure = surface_pressure  # bars
        self.has_ring_system = has_ring_system
        self.has_global_magnetic_field = has_global_magnetic_field

        # Moons
        self.moon_objects = []

    2 usages (1 dynamic)
    def update_position(self, time_scale=1.0):
        """Update planet position based on orbital motion"""
        self.angle += self.orbital_speed * time_scale
```

```
        self.rotation_angle += 0.02 * time_scale  # Planet rotation

        # Calculate position in 3D space with inclination
        self.x = self.distance_from_sun * math.cos(self.angle)
        self.y = self.distance_from_sun * math.sin(self.angle) * math.cos(math.radians(self.orbital_inclination))
        self.z = self.distance_from_sun * math.sin(self.angle) * math.sin(math.radians(self.orbital_inclination))

        # Update moon positions
        for moon in self.moon_objects:
            moon.update_position(time_scale)

    2 usages (1 dynamic)
    def reset_position(self):
        """Reset planet to initial position"""
        self.angle = self.initial_angle
        self.rotation_angle = 0
        # Reset moon positions
        for moon in self.moon_objects:
            moon.reset_position()


1 usage
class Moon:
    """Class to represent a moon orbiting a planet"""

    def __init__(self, name, planet, radius, distance, orbital_speed, color, facts=None):
        self.name = name
        self.planet = planet
        self.radius = radius
        self.distance = distance
        self.orbital_speed = orbital_speed
        self.color = color
        self.facts = facts or f"Natural satellite of {planet.name}"
        self.angle = random.uniform( a: 0, 2 * math.pi)
        self.initial_angle = self.angle  # Store initial angle for reset
        self.x = 0
        self.y = 0

    1 usage (1 dynamic)
    def update_position(self, time_scale=1.0):
```

```python
    1 usage (1 dynamic)
    def update_position(self, time_scale=1.0):
        """Update moon position relative to its planet"""
        self.angle += self.orbital_speed * time_scale
        self.x = self.planet.x + self.distance * math.cos(self.angle)
        self.y = self.planet.y + self.distance * math.sin(self.angle)

    1 usage (1 dynamic)
    def reset_position(self):
        """Reset moon to initial position"""
        self.angle = self.initial_angle


1 usage
class SolarSystemExplorer:
    4 usages
    def mix_colors(self, color1, color2, ratio):
        """Mix two colors with a given ratio (0-1)"""

        # Convert hex to RGB
        def hex_to_rgb(hex_color):
            hex_color = hex_color.lstrip('#')
            return tuple(int(hex_color[i:i + 2], 16) for i in (0, 2, 4))

        # Convert RGB to hex
        def rgb_to_hex(rgb):
            return '#%02x%02x%02x' % rgb

        rgb1 = hex_to_rgb(color1)
        rgb2 = hex_to_rgb(color2)

        # Mix colors
        mixed = tuple(int(rgb1[i] * ratio + rgb2[i] * (1 - ratio)) for i in range(3))
        return rgb_to_hex(mixed)

    def __init__(self, root):
        self.root = root
        self.root.title("3D Solar System Explorer")
        self.root.geometry("1400x900")
        self.root.configure(bg='#121212')

```

```python
    1 usage
    def load_planet_data(self, filename):
        """Load planet data from CSV file"""
        try:
            with open(filename, newline='') as csvfile:
                return {row['planet']: row for row in csv.DictReader(csvfile)}
        except Exception as e:
            print(f"Error loading planet data: {e}")
            return {}

    1 usage
    def load_satellite_data(self, filename):
        """Load satellite data from CSV file"""
        try:
            with open(filename, newline='') as csvfile:
                return list(csv.DictReader(csvfile))
        except Exception as e:
            print(f"Error loading satellite data: {e}")
            return []

    1 usage
    def initialize_planets(self):
        """Initialize all planets with their properties"""
        planets_data = {
            'Mercury': {
                'radius': 5, 'distance': 80, 'speed': 0.04, 'color': '#8C7853',
                'image': 'mercury.jpg', 'axial_tilt': 0.034, 'inclination': 7.0,
                'facts': (
                    "Closest to the Sun. Temperature varies from 800°F to -300°F! "
                    "Has a very thin exosphere. Mercury's year is just 88 Earth days."
                )
            },
            'Venus': {
                'radius': 8, 'distance': 110, 'speed': 0.03, 'color': '#FFC649',
                'image': 'venus.jpg', 'axial_tilt': 177.4, 'inclination': 3.4,
                'facts': (
                    "Hottest planet with surface temperatures over 450°C. "
                    "Spins backward and has a thick, toxic atmosphere of CO₂ with clouds of sulfuric acid."
                )
            },
            'Earth': {
```

```python
1 usage
def count_moons(self, planet_name):
    """Count moons for a specific planet"""
    return sum(1 for moon in self.satellite_data if moon['planet'] == planet_name)

1 usage
def initialize_moons(self):
    """Initialize moon objects for each planet"""
    moon_data = {
        'Earth': [('Moon', 3, 20, 0.05, '#AAAAAA')],
        'Mars': [('Phobos', 1, 15, 0.1, '#BBBBBB'), ('Deimos', 1, 18, 0.08, '#CCCCCC')],
        'Jupiter': [('Io', 2, 30, 0.15, '#FFE4B5'), ('Europa', 2, 35, 0.12, '#FFF8DC'),
                    ('Ganymede', 3, 40, 0.1, '#F5DEB3'), ('Callisto', 2, 45, 0.08, '#D2B48C')],
        'Saturn': [('Titan', 3, 40, 0.07, '#FFA07A'), ('Rhea', 2, 35, 0.09, '#F0E68C'),
                   ('Iapetus', 2, 50, 0.06, '#CD853F')],
        'Uranus': [('Titania', 2, 30, 0.05, '#ADD8E6'), ('Oberon', 2, 35, 0.045, '#87CEFA')],
        'Neptune': [('Triton', 2, 30, 0.04, '#AFEEEE')],
        'Pluto': [('Charon', 1, 15, 0.03, '#D3D3D3')]
    }

    for planet in self.planets:
        if planet.name in moon_data:
            for moon_info in moon_data[planet.name]:
                name, radius, distance, speed, color = moon_info
                moon = Moon(name, planet, radius, distance, speed, color)
                planet.moon_objects.append(moon)

1 usage
def load_images(self):
    """Load planet and ring images"""
    # Load Sun image
    sun_paths = ['images/sun.jpg', 'images/sun.png', 'images/Sun.jpg', 'images/Sun.png']
    for path in sun_paths:
        if os.path.exists(path):
            try:
                img = Image.open(path)
                img = img.resize( size: (100, 100), Image.Resampling.LANCZOS)
                self.sun_image = ImageTk.PhotoImage(img)
                break
            except Exception as e:
                print(f"Error loading sun image: {e}")
```

```python
                    print(f"Error loading ring image: {e}")

    1 usage
    def create_texture_map(self, img, base_color):
        """Create a texture map for 3D effect with lighting"""
        try:
            # Convert to RGBA if not already
            if img.mode != 'RGBA':
                img = img.convert('RGBA')

            # Create a lighting effect
            width, height = img.size
            pixels = img.load()

            for y in range(height):
                for x in range(width):
                    r, g, b, a = pixels[x, y]
                    # Apply lighting based on position (simulate sphere)
                    dx = (x - width / 2) / (width / 2)
                    dy = (y - height / 2) / (height / 2)
                    distance = math.sqrt(dx * dx + dy * dy)

                    if distance <= 1.0:  # Inside circle
                        # Lighting intensity (simulate 3D sphere)
                        intensity = 1.0 - distance * 0.7
                        intensity = max(0.3, min(1.0, intensity))
                        # Adjust color
                        r = int(r * intensity)
                        g = int(g * intensity)
                        b = int(b * intensity)
                        # Keep original alpha
                        pixels[x, y] = (r, g, b, a)
                    else:
                        # Transparent outside
                        pixels[x, y] = (0, 0, 0, 0)

            return img
        except Exception as e:
            print(f"Error creating texture map: {e}")
            return None
```

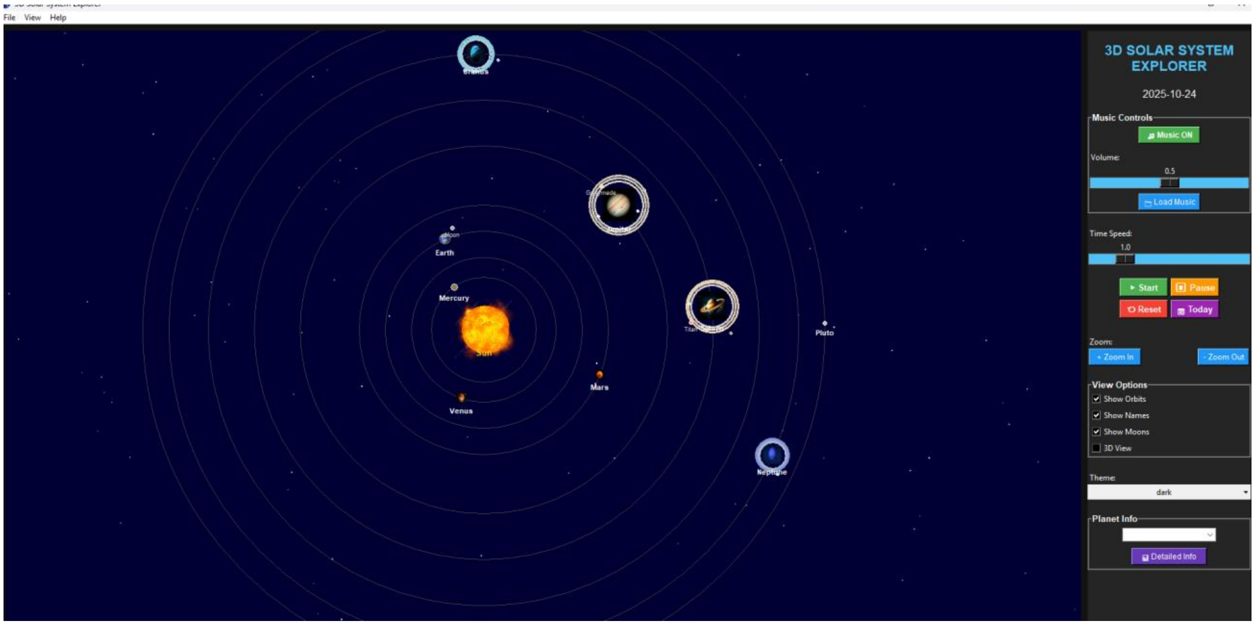
main.py solar_app.py

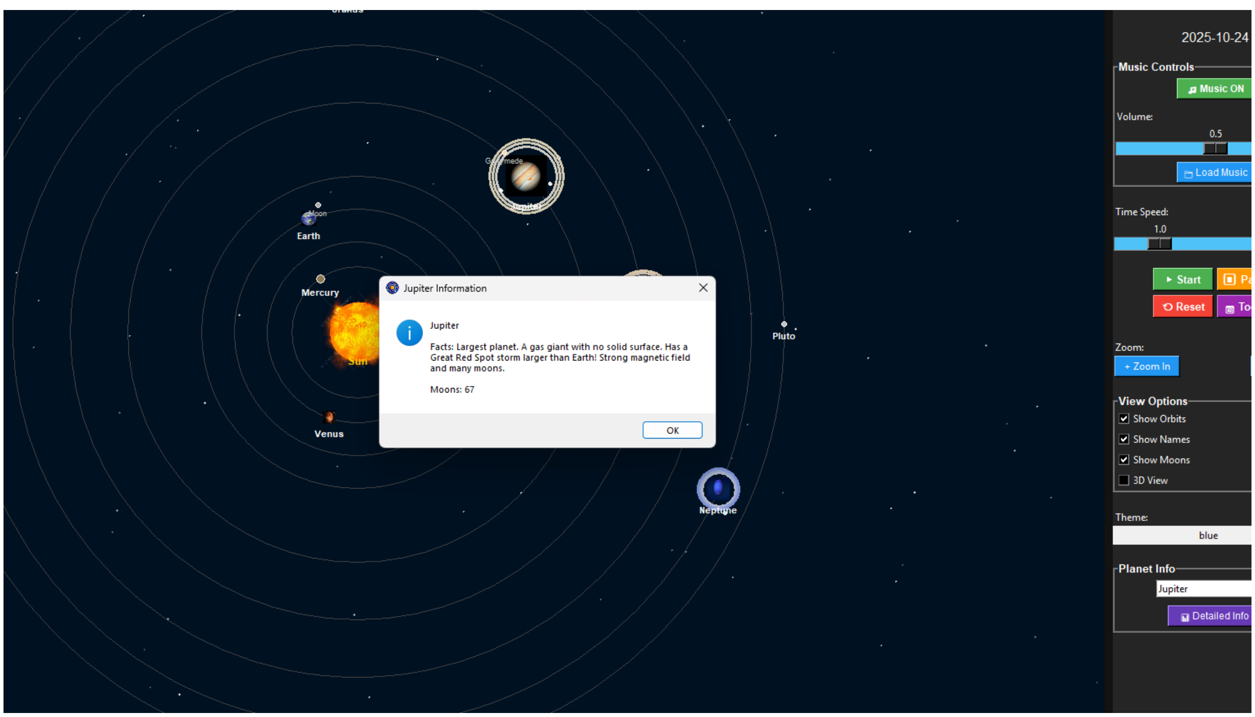
```python
class SolarSystemExplorer:
    def on_canvas_drag(self, event):
            self.center_y += dy
            self.last_x = event.x
            self.last_y = event.y
            self.draw_solar_system()

    1 usage
    def on_mouse_wheel(self, event):
        """Handle mouse wheel for zooming"""
        if event.delta > 0:
            self.zoom_factor *= 1.1
        else:
            self.zoom_factor /= 1.1
        self.draw_solar_system()

    1 usage
    def on_mouse_move(self, event):
        """Handle mouse movement for hover effects"""
        planet = self.get_planet_at_position(event.x, event.y)
        if planet:
            self.canvas.config(cursor="hand2")
        else:
            self.canvas.config(cursor="")

    2 usages
    def get_planet_at_position(self, x, y):
        """Get planet at given canvas position"""
        for planet in self.planets:
            px = self.center_x + planet.x * self.zoom_factor
            py = self.center_y + planet.y * self.zoom_factor
            if self.show_3d:
                py += planet.z * self.depth_factor * self.zoom_factor
            distance = math.sqrt((x - px) ** 2 + (y - py) ** 2)
            if distance <= planet.radius * self.zoom_factor:
                return planet
        return None

    1 usage
    def animate(self):
        """Main animation loop"""
        while self.is_running:
```

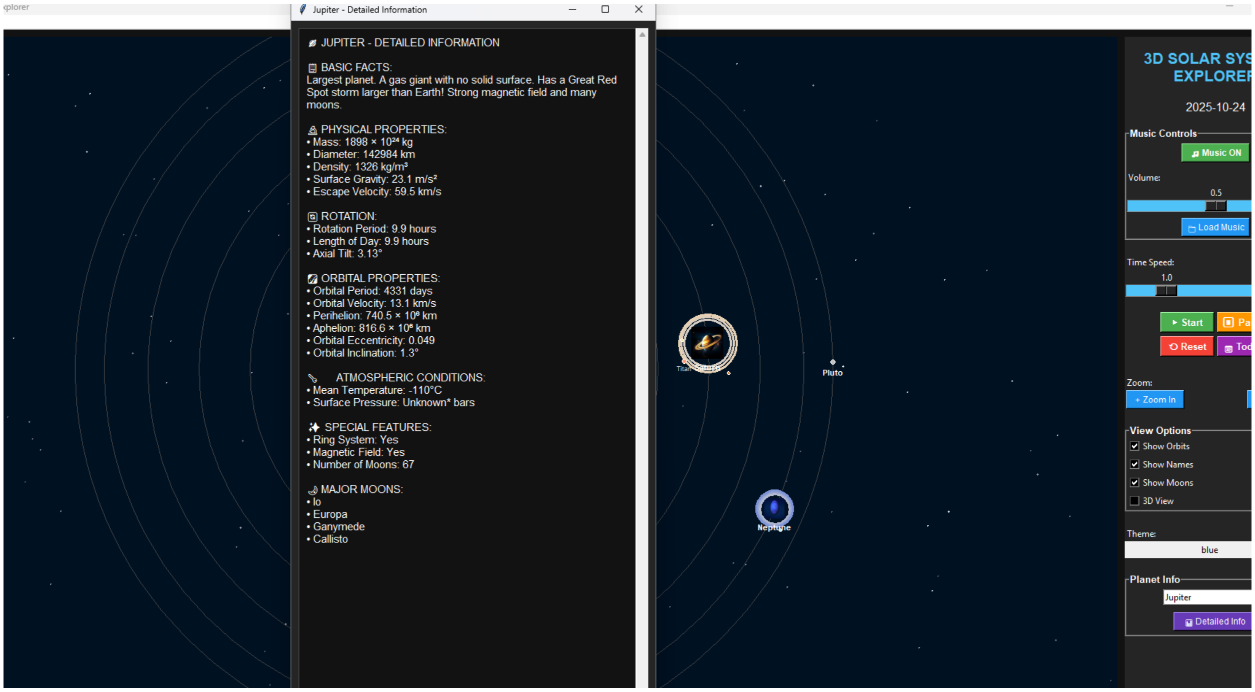
main.py solar_app.py

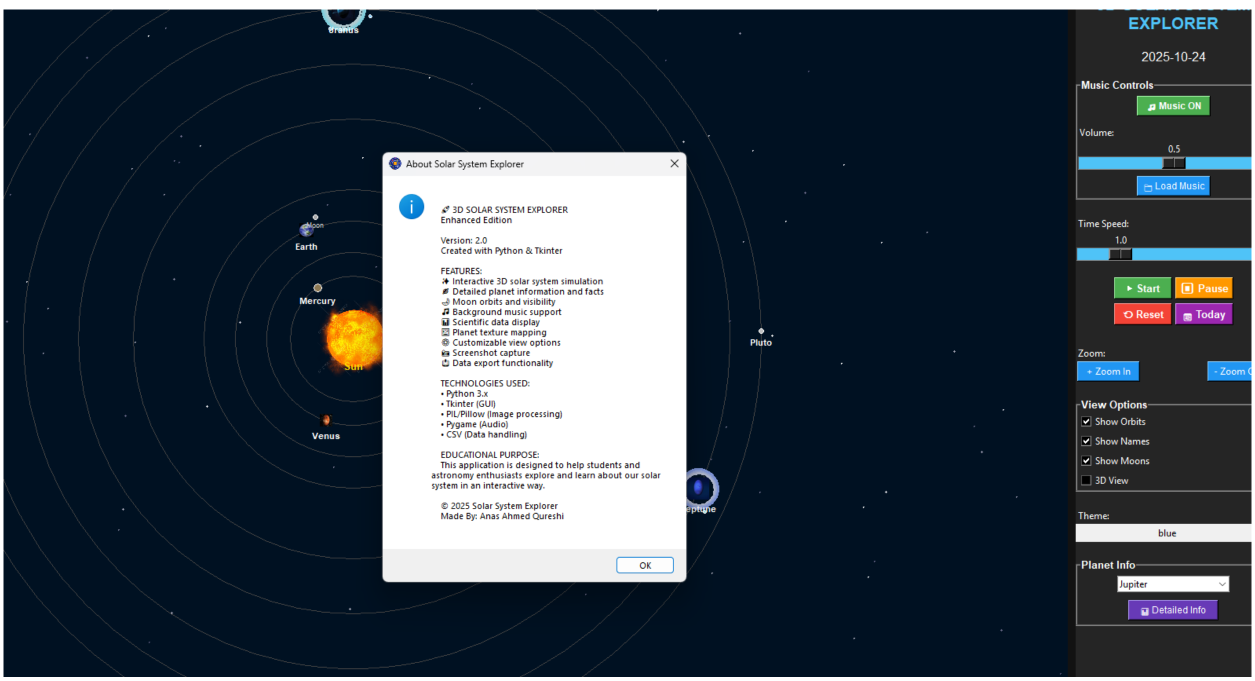
```python

1 usage
def print_hi(name):
    print(f'Hi, {name}')


if __name__ == '__main__':
    print_hi('My name is Anas Ahmed Qureshi and this is my final project in Codex Computing.'
             'The 3D Solar System Explorer is a Python-based educational simulation that visualizes our solar system in an interactive, animated 3D-like interface. '
             'The project is developed using Tkinter, Python's built-in GUI library, and enhanced with advanced mathematical modeling to simulate orbital dynamics and perspective-based 3D visua

```

16. Output Snippets






Jupiter - Detailed Information
JUPITER - DETAILED INFORMATION
BASIC FACTS:
Largest planet. A gas giant with no solid surface. Has a Great Red Spot storm larger than Earth! Strong magnetic field and many moons.
PHYSICAL PROPERTIES:
• Mass: 1898 × 10²⁴ kg
• Diameter: 142984 km
• Density: 1326 kg/m³
• Surface Gravity: 23.1 m/s²
• Escape Velocity: 59.5 km/s
ROTATION:
• Rotation Period: 9.9 hours
• Length of Day: 9.9 hours
• Axial Tilt: 3.13°
ORBITAL PROPERTIES:
• Orbital Period: 4331 days
• Orbital Velocity: 13.1 km/s
• Perihelion: 740.5 × 10⁶ km
• Aphelion: 816.6 × 10⁶ km
• Orbital Eccentricity: 0.049
• Orbital Inclination: 1.3°
ATMOSPHERIC CONDITIONS:
• Mean Temperature: -110°C
• Surface Pressure: Unknown* bars
SPECIAL FEATURES:
• Ring System: Yes
• Magnetic Field: Yes
• Number of Moons: 67
MAJOR MOONS:
• Io
• Europa
• Ganymede
• Callisto
Pluto
Neptune
3D SOLAR SYS
EXPLORER
2025-10-24
Music Controls
Music ON
Volume:
0.5
Load Music
Time Speed:
1.0
Start
Reset
Zoom:
+ Zoom In
View Options
Show Orbits
Show Names
Show Moons
3D View
Theme:
blue
Planet Info
Jupiter
Detailed Info


Uranus
Moon
Earth
Mercury
Sun
Venus
Pluto
Neptune
About Solar System Explorer
3D SOLAR SYSTEM EXPLORER
Enhanced Edition
Version: 2.0
Created with Python & Tkinter
FEATURES:
Interactive 3D solar system simulation
Detailed planet information and facts
Moon orbits and visibility
Background music support
Scientific data display
Planet texture mapping
Customizable view options
Screenshot capture
Data export functionality
TECHNOLOGIES USED:
• Python 3.x
• Tkinter (GUI)
• PIL/Pillow (Image processing)
• Pygame (Audio)
• CSV (Data handling)
EDUCATIONAL PURPOSE:
This application is designed to help students and astronomy enthusiasts explore and learn about our solar system in an interactive way.
© 2025 Solar System Explorer
Made By: Anas Ahmed Qureshi
OK
EXPLORER
2025-10-24
Music Controls
Music ON
Volume:
0.5
Load Music
Time Speed:
1.0
Start
Pause
Reset
Today
Zoom:
+ Zoom In
View Options
Show Orbits
Show Names
Show Moons
3D View
Theme:
blue
Planet Info
Jupiter
Detailed Info