**Project Report**

Zahra Batool

W25\_PAK\_INP\_AI\_27

**Solar System Simulation**

**22-12-2024**

**Syed Kumail Haider**

Instructor

**Introduction**

The Solar System simulation project is a graphical representation of the motion of celestial bodies around the Sun. Using Python and the matplotlib library, the simulation animates the orbital motion of planets to provide an educational and visual understanding of planetary dynamics. This project models the Sun and eight planets (Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, and Neptune) using simplified orbital mechanics principles. The system visualizes the relative distances and motion of planets in a 2D plane.

**Objectives**

The main objectives of this project are:

1. To create a visually engaging simulation of the Solar System.
2. To model the orbital motion of celestial bodies using basic trigonometric functions.
3. To provide an educational tool for understanding planetary motion and relative distances.
4. To explore the application of Python libraries for scientific visualization.

**Methodology**

**Tools and Libraries**

* **Python**: Programming language used to implement the simulation.
* **Matplotlib**: Library for plotting and animating the celestial bodies.
* **Numpy**: Used for mathematical calculations such as trigonometric functions.

**Design Approach**

**a) CelestialBody Class**

The CelestialBody class defines the properties and behaviors of each celestial body.

* **Attributes**:
  + name: Name of the celestial body.
  + distance: Orbital distance from the Sun (in Astronomical Units).
  + radius: Physical radius of the body (for additional detail, not used in the visualization).
  + color: Color for graphical representation.
  + orbital\_speed: Orbital angular velocity (degrees per day).
  + angle: Initial position in its orbit (default is 0).
* **Methods**:
  + update\_position(time): Updates the angular position based on orbital speed and elapsed time.
  + get\_position(): Calculates the Cartesian coordinates (x, y) of the celestial body.

**b) SolarSystem Class**

The SolarSystem class handles the collective simulation of celestial bodies.

* **Attributes**:
  + bodies: A list of celestial bodies.
* **Methods**:
  + add\_body(body): Adds a celestial body to the Solar System.
  + simulate(duration, time\_step): Runs the animation for a specified duration and time step.

**Implementation**

**Initialization**

A SolarSystem object is initialized. Celestial bodies are added using the add\_body() method, with parameters specifying their distance, radius, color, and orbital speed.

**Simulation**

The simulate() method animates the Solar System:

* A loop iterates over the specified time duration in increments of time\_step.
* For each time step, the update\_position() method recalculates the angular position of each celestial body.
* Using get\_position(), the new Cartesian coordinates are calculated and plotted.
* The Sun is plotted at the origin, and the planets are plotted at their respective positions with dashed lines connecting them to the Sun.
* The matplotlib animation is updated frame-by-frame using plt.pause().

**Results**

**Visualization**

* The Sun is represented as a yellow circle at the center.
* Planets are shown as colored dots moving in circular orbits around the Sun.
* The simulation accurately represents relative speeds, with Mercury completing its orbit faster than Neptune.
* Dashed lines visually connect each planet to the Sun, highlighting their paths.

**Animation**

* The animation provides a dynamic representation of planetary motion over a year.
* Orbital speeds are scaled to demonstrate the approximate time each planet takes to complete one revolution around the Sun.

**Key Features and Challenges**

**Features**

* Simplified yet accurate depiction of planetary orbits.
* Interactive and visually engaging animation.
* Modular code structure for easy addition of more celestial bodies.

**Challenges**

* Approximations were made for orbital speeds and distances to maintain simplicity.
* The 2D visualization omits the tilt and elliptical nature of real orbits.
* Computational constraints limit the ability to include moons or other smaller celestial bodies.

**Applications**

* **Educational Tool**: Provides a simplified visualization of planetary motion.
* **Data Visualization**: Demonstrates the use of Python libraries for dynamic data representation.
* **Foundation for Advanced Models**: Can be extended to include more complex dynamics like elliptical orbits, axial tilts, or gravitational interactions.

**Conclusion**

The Solar System simulation successfully visualizes the orbital motion of planets in a 2D representation. It demonstrates the potential of Python for scientific and educational applications. While the model simplifies real-world dynamics, it provides an engaging platform for understanding the basics of planetary motion.

Future enhancements could include:

* Modeling elliptical orbits.
* Adding moons and asteroids.
* Including interactive features for user-defined simulations.