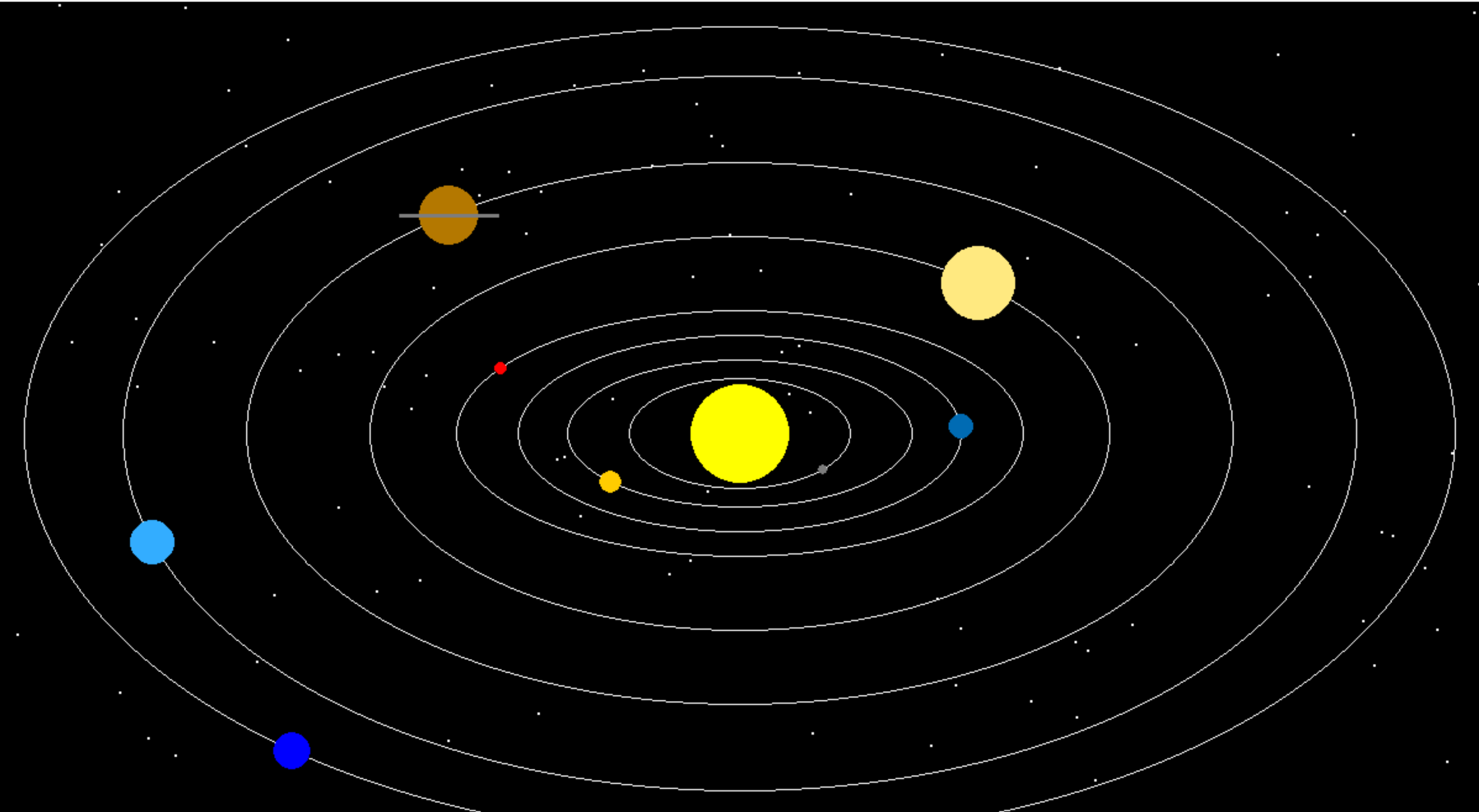
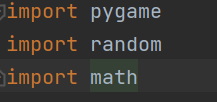
Computational Physics

Solar System Simulation



Explanation for the project:



These import the necessary libraries.

pygame: Used for creating the graphical window and drawing the objects.

random: Used to generate random positions for the stars.

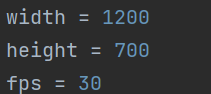
math: Provides mathematical functions, such as cos and sin for calculating planet positions.

Initialization:



This initializes all the Pygame modules.

Constants:

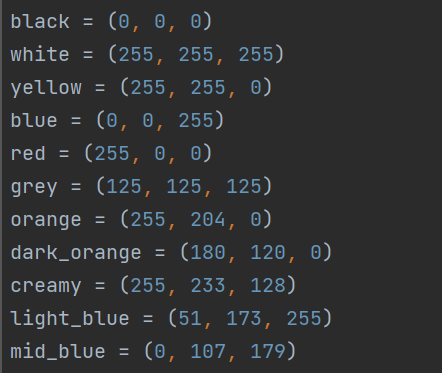


These constants define the dimensions of the window and the frame rate.

WIDTH and HEIGHT set the size of the window.

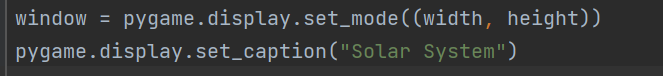
FPS sets the frames per second for the game loop, controlling the update speed.

Colors :



These tuples define RGB values for various colors used in the simulation.

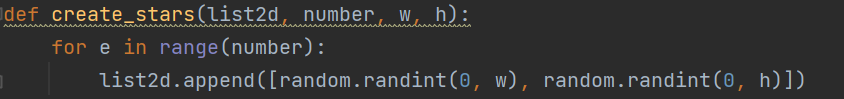
Setup Display :



This sets up the Pygame window with the specified width and height, and sets the window title to "Solar System".

Functions:

Creating and Drawing Stars:

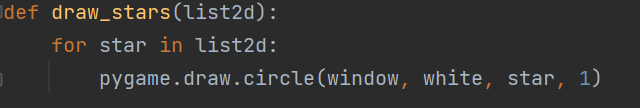


This function populates a list with randomly positioned stars.

stars\_list: The list to be filled with star coordinates.

number: The number of stars to create.

w and h: The width and height boundaries within which the stars are placed.

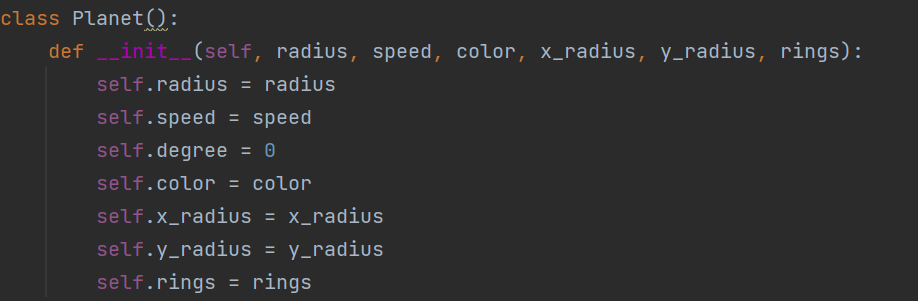


This function draws the stars on the window.

Each star is drawn as a small circle (radius 1) using the coordinates in stars\_list.

Class Definitions:

Planet Class:



The Planet class represents a planet in the solar system.

radius: Radius of the planet.

speed: Orbital speed of the planet.

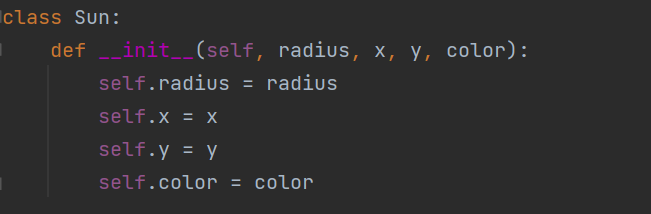
degree: Current degree position in orbit (initially set to 0).

color: Color of the planet.

x\_radius and y\_radius: Radii of the elliptical orbit.

rings: Boolean indicating if the planet has rings.

Sun Class:

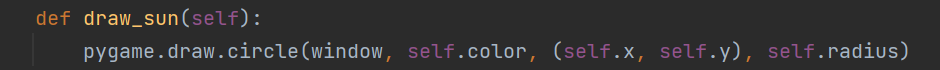


The Sun class represents the sun.

radius: Radius of the sun.

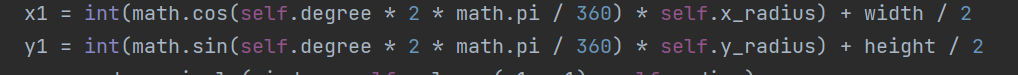
x and y: Coordinates of the sun's center.

color: Color of the sun.



The draw method draws the sun as a circle at its specified position.

Calculating the Planet's Position:



The formula used in the Planet class's draw method is based on the parametric equations of an ellipse.

x = int(math.cos(math.radians(self.degree)) \* self.x\_radius) + WIDTH // 2

y = int(math.sin(math.radians(self.degree)) \* self.y\_radius) + HEIGHT // 2

1. **Convert Degrees to Radians:**
   * math.radians(self.degree) converts the current angle from degrees to radians because trigonometric functions in Python's math module expect angles in radians.
2. **Calculate X and Y Coordinates:**
   * math.cos(math.radians(self.degree)) \* self.x\_radius: Computes the x-coordinate based on the cosine of the angle, scaled by the semi-major axis (self.x\_radius) of the ellipse.
   * math.sin(math.radians(self.degree)) \* self.y\_radius: Computes the y-coordinate based on the sine of the angle, scaled by the semi-minor axis (self.y\_radius) of the ellipse.
3. **Translate to Screen Coordinates:**
   * + WIDTH // 2 and + HEIGHT // 2: Adjusts the coordinates so that the center of the ellipse is positioned at the center of the window.

Parametric Equations of an Ellipse:

The general parametric equations for an ellipse centered at the origin with a semi-major axis a and semi-minor axis b are:

x=a⋅cos⁡(θ)x = a \cdot \cos(\theta)x=a⋅cos(θ)

y=b⋅sin⁡(θ)y = b \cdot \sin(\theta)y=b⋅sin(θ)

Where θ\thetaθ is the angle in radians.

In the code:

self.x\_radius corresponds to the semi-major axis (a).

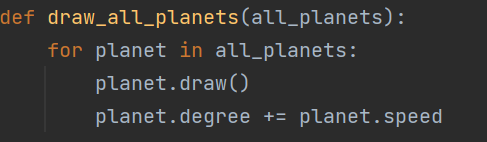
self.y\_radius corresponds to the semi-minor axis (b).

self.degree is the angle in degrees, which is converted to radians using math.radians(self.degree).

The position of each planet is calculated using the parametric equations for an ellipse, with the center of the ellipse translated to the center of the screen. The trigonometric functions cos and sin are used to compute the coordinates based on the current angle, which is updated incrementally to animate the orbits.

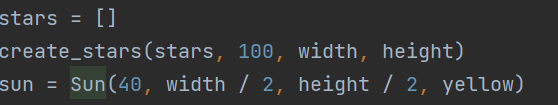
Updating the Planet's Position:

Each planet's position is updated in the draw\_all\_planets function by increasing its angle (self.degree) by its speed (self.speed)



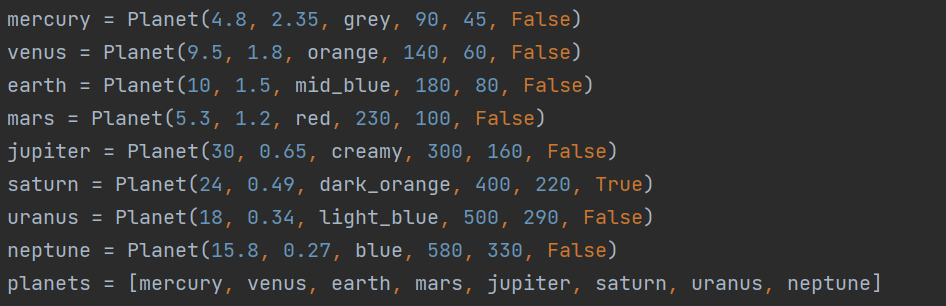
planet.degree += planet.speed increments the angle of the planet, causing it to move along its orbit in the next frame.

Creating Stars and Sun:



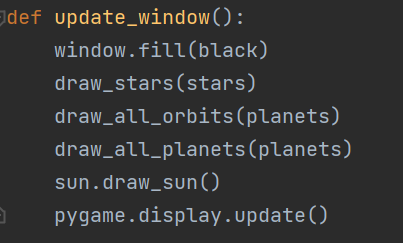
This creates a list of 100 stars and an instance of the Sun class.

Creating Planets:



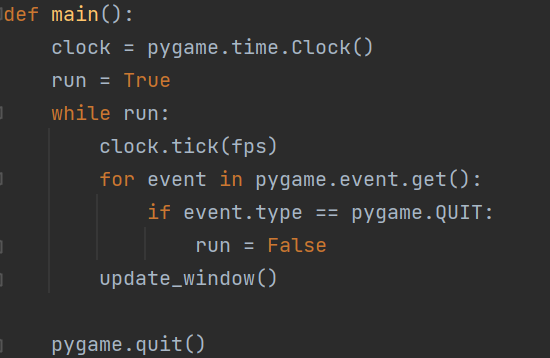
This creates instances of the Planet class for each planet in the solar system with their respective properties.

Function to Update the Window:



This function clears the window, draws the stars, orbits, planets, and the sun, and updates the display.

Main Function:



The main function contains the game loop.

clock is used to control the frame rate.

The loop runs until the window is closed (run is False).

clock.tick(FPS) ensures the loop runs at the defined frame rate.

pygame.event.get() processes events, and if the QUIT event is detected, it stops the loop.

update\_window() updates the window each frame.

pygame.quit() cleans up and closes the window when the loop ends.

Run the Main Function:



This condition checks if the script is being run directly (not imported) and calls the main function to start the simulation.

This code sets up a basic solar system simulation with stars, planets orbiting around the sun, and basic graphical elements. Each planet's position updates based on its speed, creating the animation effect.

**Project 2**

Introduction

The Solar System has captivated human curiosity for centuries, with its celestial bodies, planetary orbits, and cosmic interactions. Understanding the vastness and complexity of our cosmic neighborhood has been a driving force behind the development of various scientific tools and technologies. One such tool is the Solar System Simulator, a virtual representation that allows us to explore and comprehend the intricacies of our planetary system.

The Solar System simulator allows users to explore and interact with a virtual representation of our solar system. It provides a dynamic and immersive experience, enabling individuals to learn about the celestial bodies that comprise our cosmic neighborhood. The concept of a Solar System Simulator emerged from the fascination humans have had with the skies above them for centuries. From the early astronomers who studied the movements of the planets to the modern space exploration missions, our curiosity about the cosmos has driven us to seek a deeper understanding of the celestial objects that surround us.

With advancements in computer technology, scientists, educators, and enthusiasts have developed various software applications to simulate the workings of the solar system. These simulators utilize complex algorithms and accurate astronomical data to create realistic representations of the Sun, planets, moons, and other celestial bodies. Solar System Simulators offer a wide range of features and functionalities. They allow users to observe the motion of celestial bodies, study their orbits, and simulate various astronomical phenomena such as eclipses, planetary alignments, and meteor showers. These simulations can be adjusted to different time scales, enabling users to observe the past, present, and future positions of celestial objects.

Problem

As we assess the current state of solar system simulators, a number of challenges and areas for improvement become apparent. Understanding these issues is crucial for further advancements in the field and ensuring the optimal user experience.

Here is a problem analysis for today's solar system simulators:

The user interface and interactivity of solar system simulators vary widely. Some simulators may lack intuitive controls or present a steep learning curve, making it difficult for users to navigate and fully engage with the simulations. Enhancements in user experience design and interface intuitiveness are necessary for better accessibility and engagement.

While solar system simulators have educational potential, their integration into formal education systems may be limited. There is often a lack of comprehensive educational content and interactive features. Simulators may provide basic information about celestial bodies, but deeper explanations, scientific context, and educational resources may be limited. Additionally, interactive features, such as the ability to manipulate variables, create custom scenarios, or simulate hypothetical situations, may be lacking, restricting the learning experience and user engagement.

To address the limitations mentioned earlier, we decided to develop a solar system simulator that allows users to input variables and plots them into the simulation. The first step in this process involved creating a GUI (Graphical User Interface) to gather user input for variables such as mass, coordinates, and velocity.

Using a module named tkinter, which provides capabilities for creating user interfaces, we were able to design an interactive interface where users can input the characteristics of the planets. These inputs are then stored in the planet variable, creating a representation of each planet within the simulation.

Upon pressing the update button, the entered inputs are appended to the planet variable, and a pop-up window appears, displaying the solar system simulator. Within this simulator, the planets are graphically plotted according to their defined characteristics.

To facilitate the creation of the user interface, we utilized the tkinter module, which offers a range of tools and functions for building interactive GUIs. This module enables us to design a visually appealing and user-friendly interface that enhances the overall user experience.

Furthermore, we leveraged the capabilities of Matplotlib, a plotting library, to visualize the planets within the solar system simulator. Matplotlib provides a flexible platform for creating various types of plots and allows us to incorporate physics calculations and simulations into our simulator.

By combining the functionality of tkinter and the plotting capabilities of Matplotlib, we have created a solar system simulator that not only takes user input but also provides a visual representation of the simulated celestial bodies. This allows users to observe and study the effects of different variables on the movement and behavior of the planets, thereby enhancing their understanding of the underlying physics principles governing our solar system.

**The reason**

To enhance the understanding of celestial body movements and educate users about the dynamics of our solar system, we plan to develop a GUI (Graphical User Interface) that allows for accurate visualization of the effects of physics. This GUI provides an interactive platform where users can manipulate the position and velocity of each body in the solar system to observe and predict their paths of travel.

By incorporating real-life physics equations into the program, we have created a simulation that accurately reflects the principles governing the motion of celestial bodies. Users can input specific parameters such as mass, position, and velocity for each celestial body, enabling them to study and analyze the resulting trajectories.

This implementation goes beyond just providing visual representations and offers a deeper understanding of how celestial bodies move through time and space. Users can explore various scenarios and observe the consequences of altering the properties of the celestial bodies in the system.

By demonstrating the application of physics equations in the simulation, we aim to educate users about the fundamental principles that govern celestial mechanics. This interactive approach allows users to actively engage with the simulation, fostering a hands-on learning experience.

Overall, our GUI-based solar system simulator incorporates the sun and physics equations to provide a comprehensive educational tool. It empowers users to explore and comprehend the intricate dynamics of our solar system, promoting a deeper appreciation for the laws of physics and the marvels of celestial motion.

Related works

The solar system simulator developed by The Sky Live (TheSkyLive.com, n.d.) provides a visually captivating representation of our solar system as it exists in the present moment. It offers a realistic depiction of the orbits of planets, asteroids, and other celestial bodies, complete with accurate labels and names. Users can observe the intricate dance of the planets and gain a deeper understanding of the layout and structure of our cosmic neighborhood.

However, one limitation of this simulator is the inability to add additional planets to the existing solar system. While it faithfully represents the current arrangement of celestial bodies, it does not allow for the exploration of hypothetical scenarios or the addition of new planets to the system. Nonetheless, the simulator serves as an excellent educational tool for studying known celestial objects and their movements.

On the other hand, the solar system simulator created by the University of Colorado Boulder (University of Colorado Boulder, n.d.) takes a more physics-focused approach. This simulator accurately incorporates the principles of physics, such as acceleration due to gravity and path deviation, to predict the motion and orbit of celestial bodies. It allows users to input various parameters such as position, mass, velocity, and distance between planetary bodies, and then observe the resulting interactions and trajectories.

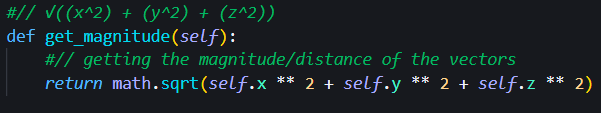
While this simulator lacks a built-in sun representation, it excels in providing a detailed analysis of how different factors influence the motion of celestial bodies. Users can experiment with various configurations and observe how changing the parameters affects the orbits and interactions between planets. This simulator is particularly valuable for those interested in studying the underlying physics of our solar system and exploring the intricate dynamics that govern celestial motion.

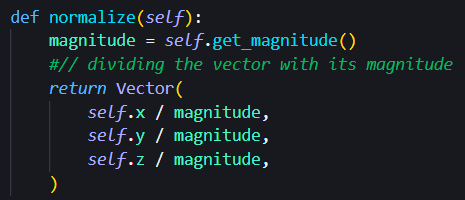
While the solar system simulator by The Sky Live focuses on providing accurate visual representations of the current solar system, the simulator developed by the University of Colorado Boulder emphasizes the physics and dynamics of celestial bodies. Each simulator offers a unique perspective and caters to different aspects of solar system exploration, enhancing our understanding and appreciation of the vastness and complexity of our cosmic neighborhood.

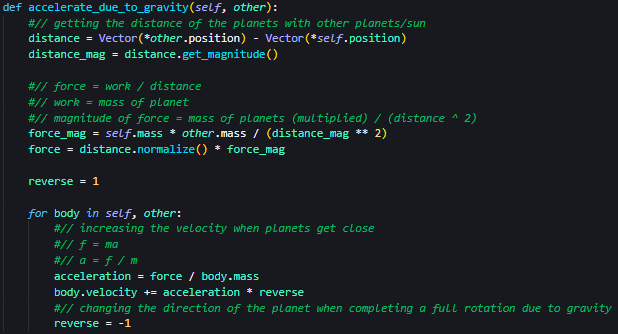
Implementation

|  |  |
| --- | --- |
| Formula | Notation |
| Distance | d = (x2+y2+z2) |
| Force basic formula | Fmag= GM1M2/d2 |
| Magnitude of Force | Fmag= (M1\*M2)/d2 |
| Force from Magnitude | F =((Fmagx/Fmag), (Fmagy/Fmag), (Fmagz/Fmag)) \* Fmag |
| Force from mass and acceleration | F = ma |
| Acceleration from force acquired and mass inputted by user | a = F/m |

Application using Python







Implementation details

We made a magnitude getter that uses the formula

distance = x2+y2+ z2

We also made a normalization function to normalize the vector with its magnitude.

The accelerate\_due\_to\_gravity() function holds the main physics formula that runs our solar system simulator. Firstly we take the distance by subtracting the planets’ coordinates from each other. Then we take force using the formula

Fmag= GM1M2/d2

To obtain work we use the mass of the planet. To obtain the magnitude of force, using the above equation we get

Fmag= (M1\*M2)/d2

Then to get the force we use the formula

F =((Fmagx/Fmag), (Fmagy/Fmag), (Fmagz/Fmag)) \* Fmag

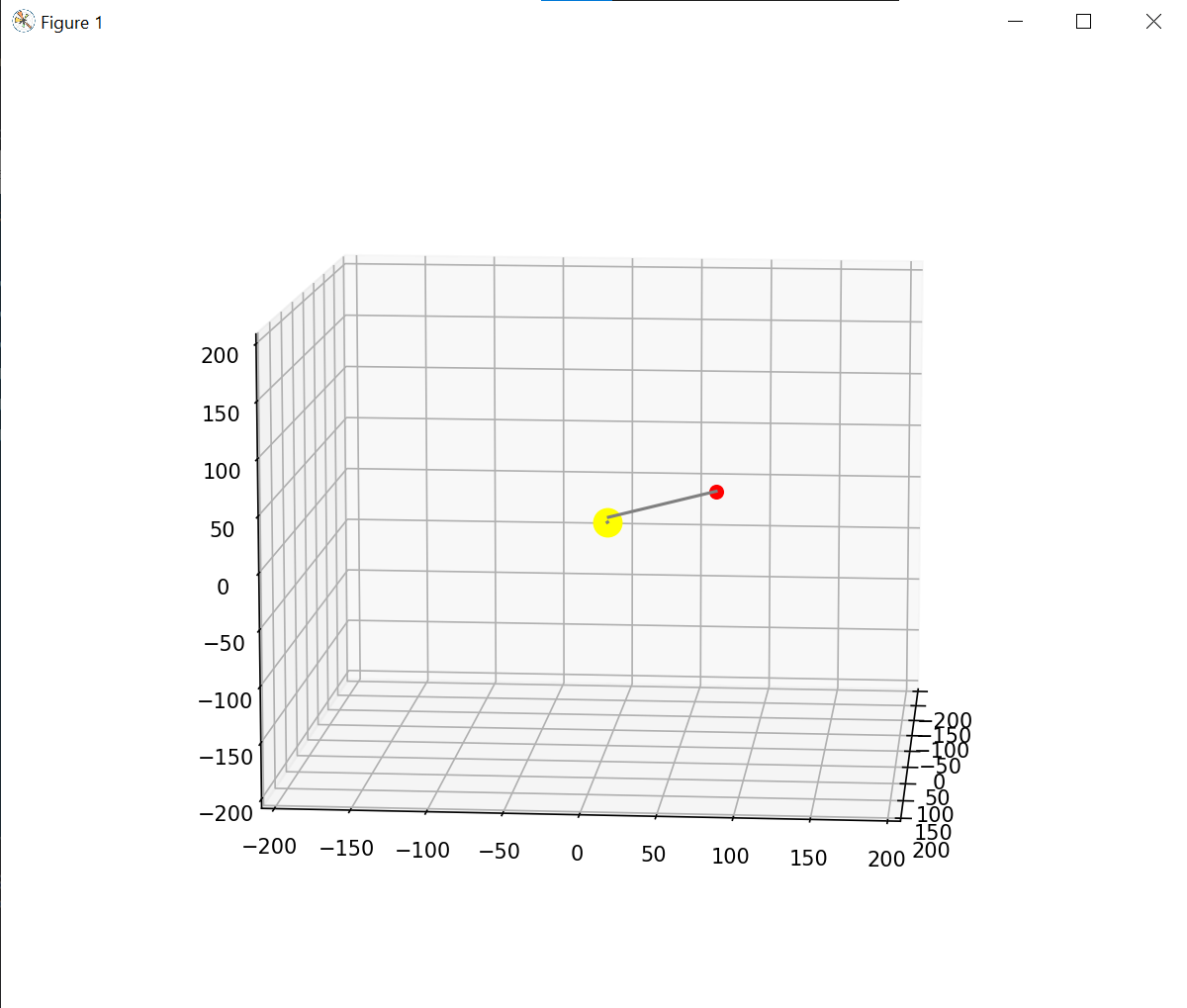
Then to obtain the acceleration we derive it from the formula

F = ma

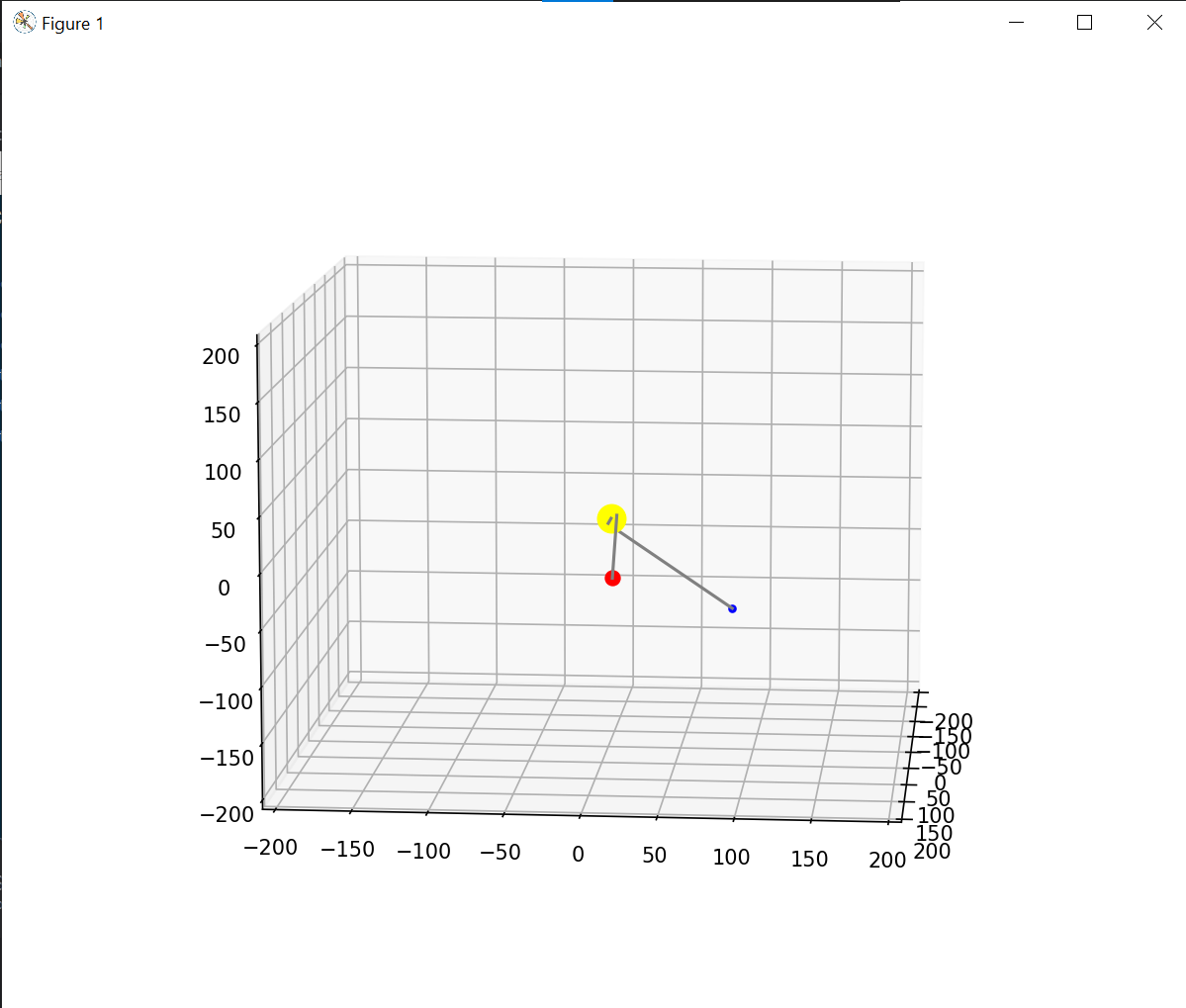
into

a = F/m.

Result



We created a body that accelerates when its position is closer to other celestial bodies. In this example, the velocity increases when the body comes close to the sun.



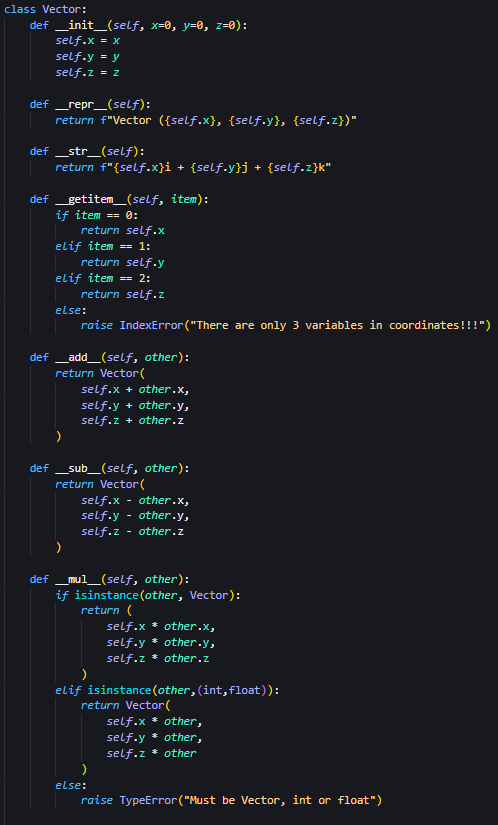
We then created another body to simulate how the velocity changes in between bodies.

Code Explanation

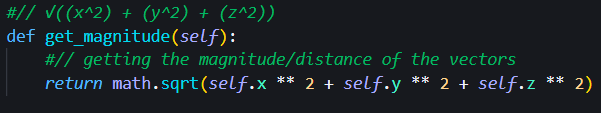




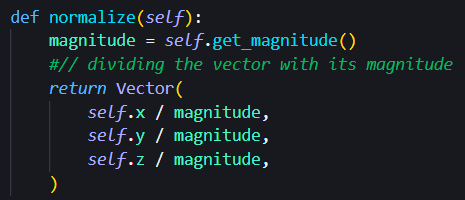
We used these modules to ease the development of our solar system simulator



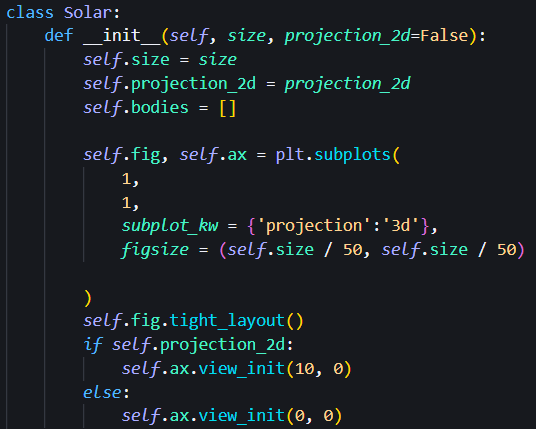
Basic vector and vector operations are made because we are not using vpython and python does not have built in vectors.



We made a magnitude getter which uses the formula magnitude = x2+y2+ z2

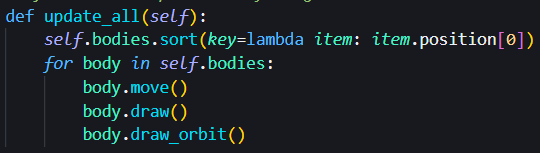


We also made a normalization function to normalize the vector with its magnitude.



The solar class is intended to make the solar system. This includes the size of our matplotlib window denoted by “*size”* which will then be inputted as our frame size.

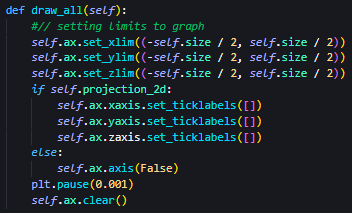




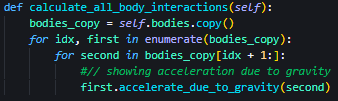
The update\_all() function is used to perform the basic visualization of our solar system simulator.

key=lambda item: item.position[0].

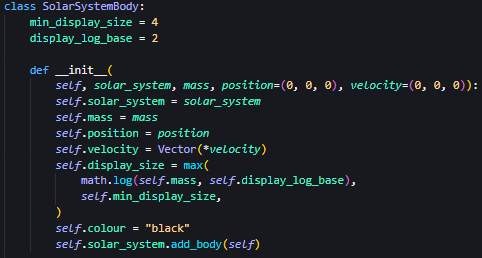
Here, a lambda function is defined to determine the key by which the sorting should be performed. In this case, it accesses the position attribute of each item in the list (item.position[0]), which suggests that the position attribute is iterable (e.g., a list in our case) and the [0] index is used as the key for sorting. To clarify, if the position attribute of each celestial body is a list or tuple representing coordinates, such as [x, y, z], this code will sort the self.bodies list based on the first element of the position attribute (i.e., the x coordinate). The resulting list will be in ascending order based on the x coordinates of the celestial bodies. This code assumes that the position attribute exists for each object in the self.bodies list and that the objects have a comparable data type for sorting to be possible.



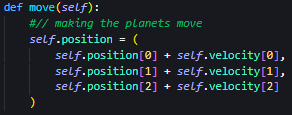
Drawing the limits of the graph.



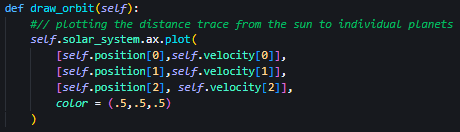
Showing the physics simulation in our solar system simulator. The accelerate\_due\_to\_gravity() function will be defined later in the code.



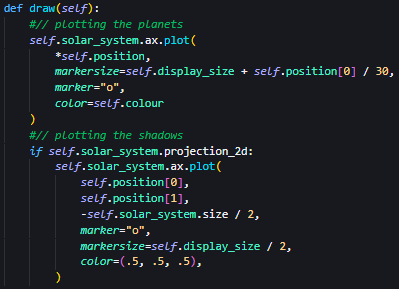
The SolarSystemBody class is intended to show the visuals in our solar system including shadows, planets and movement of the planets.



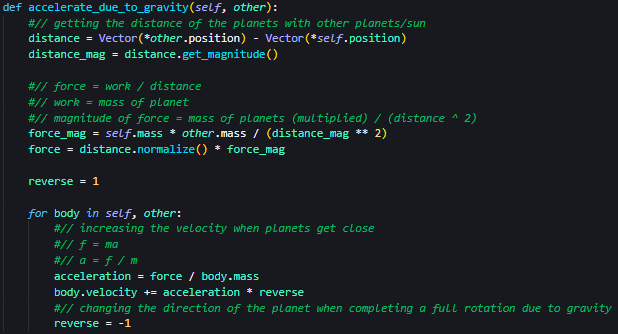
The move() function increases the position of the planet by the velocity.



The draw\_orbit() function shows the distance tracer of the planet from (0,0,0) of the graph.



The draw() function plots the planets into the graph. Projection\_2d plots the shadows of the planets.



The accelerate\_due\_to\_gravity() function holds the main physics formula that runs our solar system simulator. Firstly we take the distance by subtracting the planets’ coordinates from each other. Then we take force using the formula

F = GM1M2/d2

To obtain work we use the planet's mass. To obtain the magnitude of force, using the above equation we get

Fmag= (M1\*M2)/d2

Then to get the force we use the formula

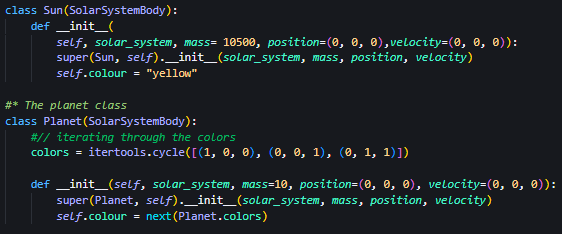
F =((Fmagx/Fmag), (Fmagy/Fmag), (Fmagz/Fmag)) \* Fmag

Then to obtain the acceleration we derive it from the formula

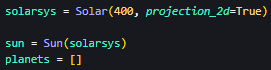
F = ma

into

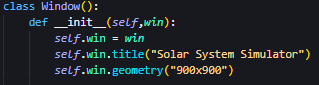
a = F/m.



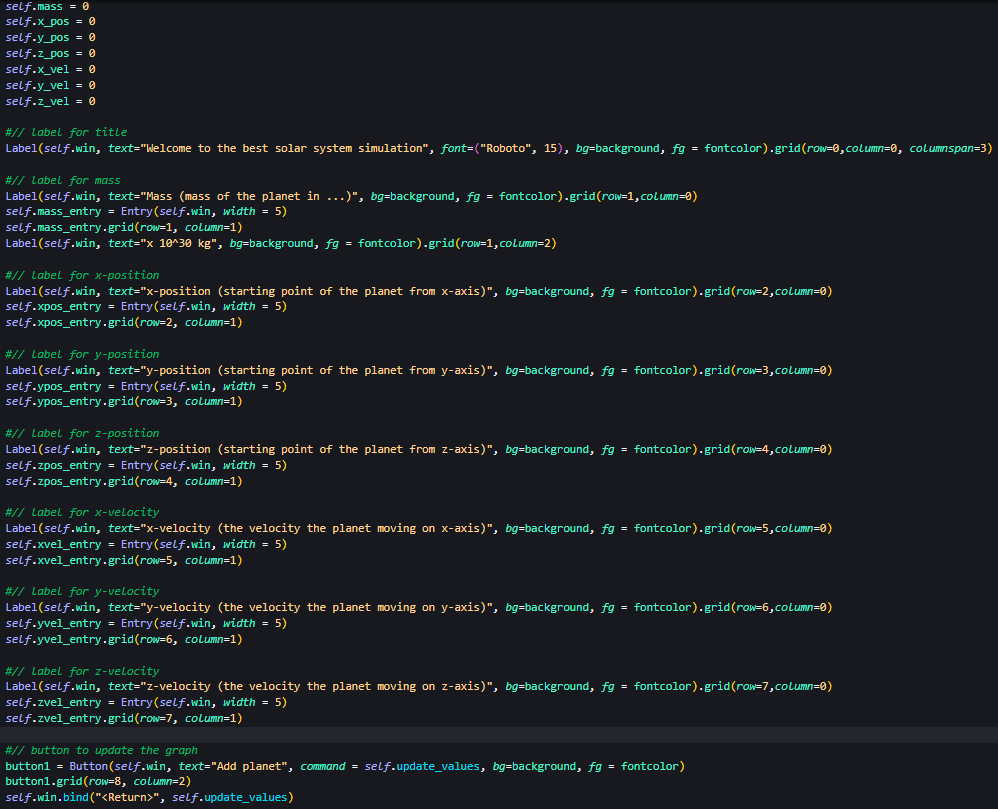
Differentiating the Sun and other planets by determining the mass and its color, the distinct yellow.



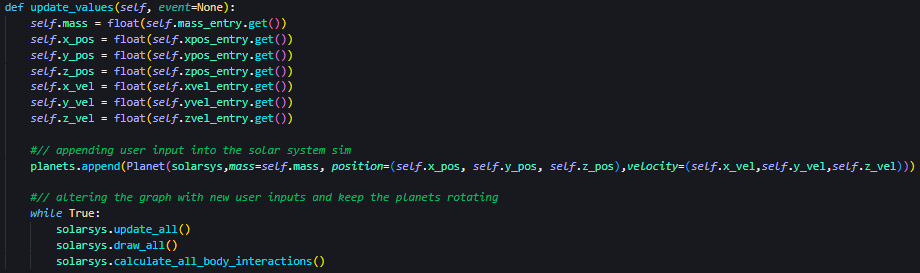
Initializing the Solar system, Sun, and planets



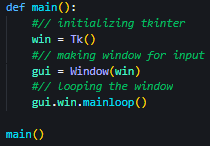
Determining the title and size of the window of the user input frame.



Basic mapping using Labels and Button from the tkinter module.



Getting user input and updating the graph by plotting the planets from a click of the ‘update’ button or ‘enter’ keyboard function.



The main() function holds all the functionalities of this solar system function by making the window for user input and then looping the window. We then call the main() function to execute the program.

Test

Planet 1: Mass = 10

Coordinates = (100,50,0)

Velocity = (0,5,5)

Planet 2: Mass = 20

Coordinates = (150,50,20)

Velocity = (0,-5,5)

Evaluation and Discussion

Upon evaluating the provided information, it is evident that the proposed solution effectively tackles the problem at hand. Several factors contribute to this evaluation.

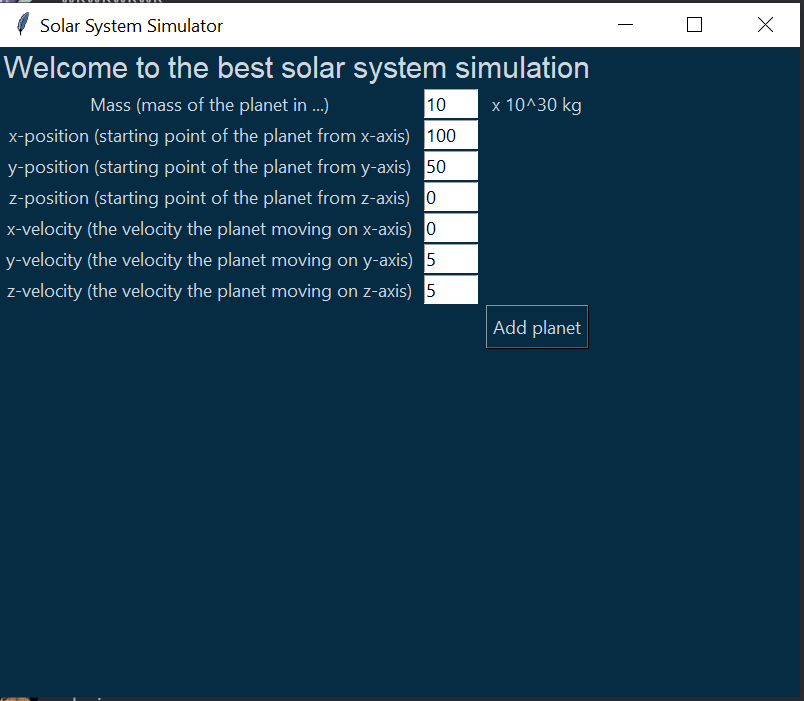
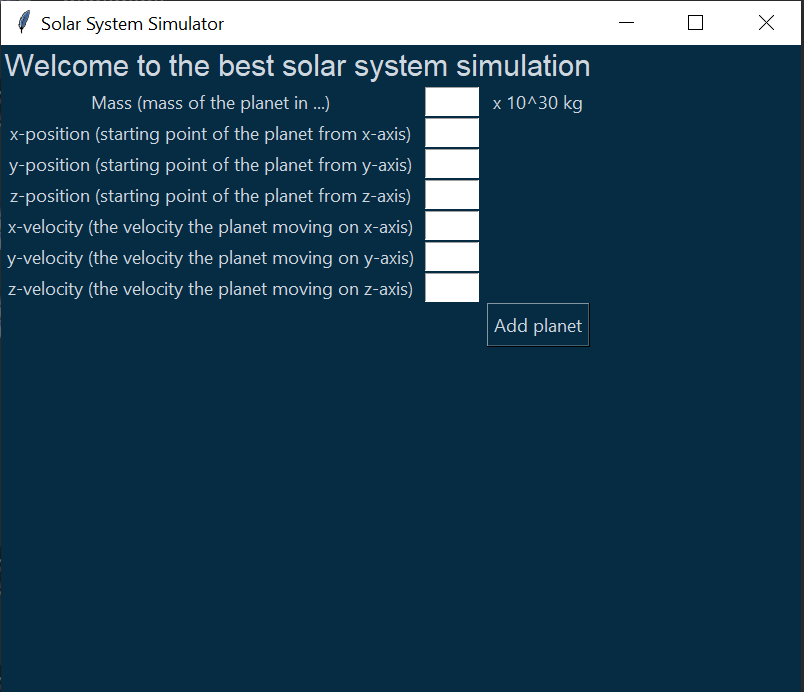
Firstly, the development of a solar system simulator demonstrates a practical approach to overcoming the limitations. By allowing users to input variables and plotting them into the simulation, the solution enables a dynamic and interactive experience. This not only enhances user engagement but also facilitates a better understanding of the subject matter.

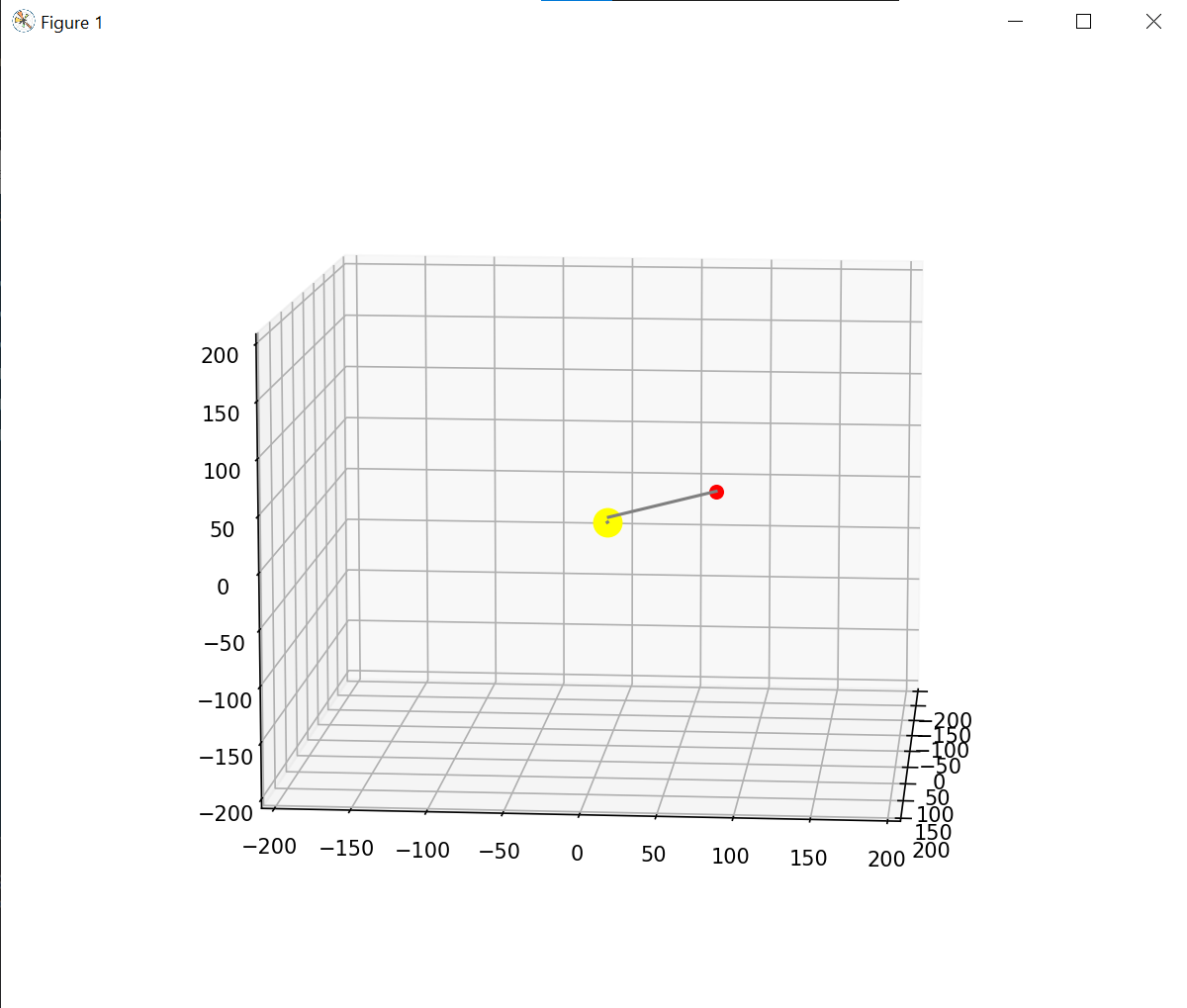
The incorporation of the tkinter module to create a user-friendly GUI is a notable aspect of the solution. By utilizing tkinter's capabilities for designing interactive interfaces, the solution ensures ease of use and accessibility for users. This enhances the overall user experience and encourages user engagement with the simulator.

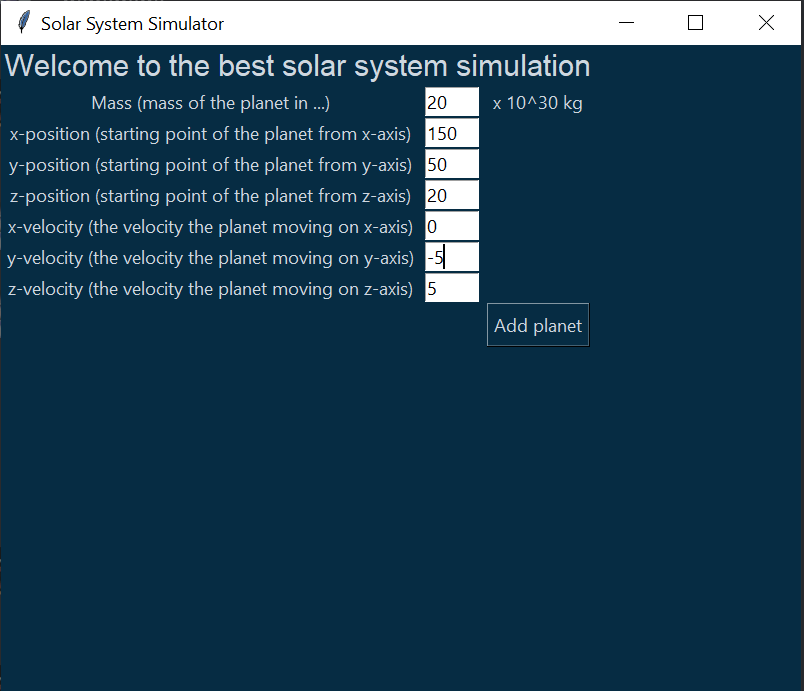
Furthermore, the use of Matplotlib, a powerful plotting library, adds value to the solution. By leveraging Matplotlib's plotting capabilities, the simulator provides a visual representation of the simulated celestial bodies. This visual aspect allows users to observe and analyze the effects of different variables on the behavior and movement of the planets. Consequently, users can gain a deeper understanding of the underlying physics principles governing our solar system.

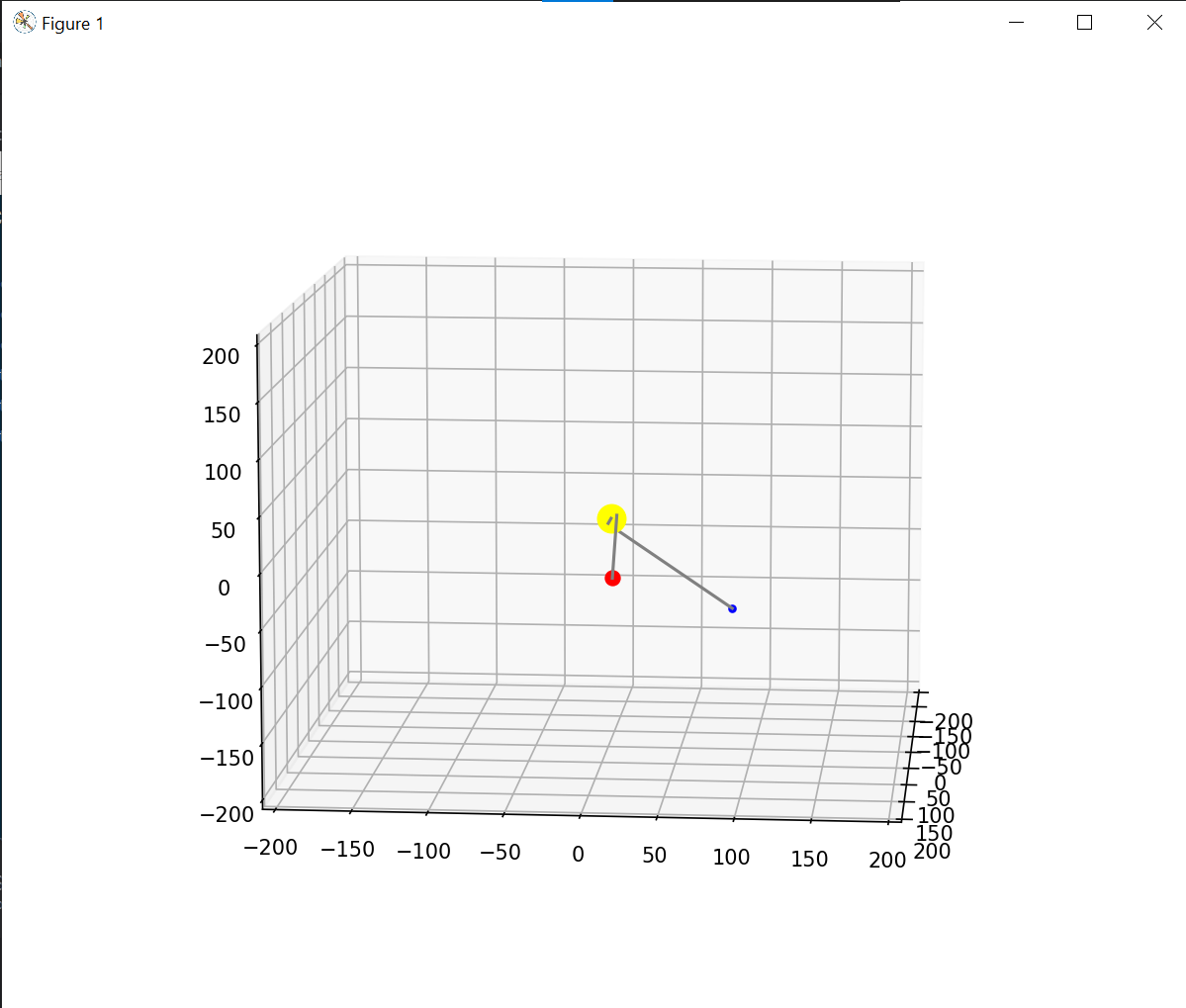
The combination of tkinter and Matplotlib creates a comprehensive solution that effectively addresses the limitations. The integration of user input, visual representation, and simulation capabilities provides a holistic approach to studying and exploring the dynamics of the solar system.

Overall, based on the information provided in the passage, the solution demonstrates a successful resolution to the stated problem. The development of a solar system simulator with an interactive GUI and visual representation showcases a comprehensive and effective approach to overcoming limitations and providing an engaging learning experience.









Conclusion and Recommendation

In conclusion, the solar system simulator provides a captivating and immersive experience that allows users to explore the vast wonders of our cosmic neighborhood. Through its intricate models and realistic simulations, it successfully captures the awe-inspiring beauty and complexity of our solar system.

By offering a user-friendly interface and a wealth of interactive features, the simulator enables users to embark on virtual journeys, deepening their understanding of celestial mechanics and astronomical phenomena. Whether it's witnessing the graceful dance of planets around the sun, observing the mesmerizing phases of the moon, or discovering the breathtaking diversity of celestial bodies, the simulator offers an educational and entertaining experience for users of all ages.

Moreover, the solar system simulator serves as a powerful educational tool, making complex astronomical concepts accessible and engaging. It allows students, educators, and enthusiasts to study the dynamics of the solar system, understand the interplay of gravity and planetary motion, and appreciate the delicate balance that sustains life on Earth.

Furthermore, the simulator's ability to incorporate real-time data and cutting-edge scientific discoveries ensures that users are kept up to date with the latest astronomical knowledge. This feature allows for a dynamic and ever-evolving experience, fostering a sense of wonder and curiosity about our place in the universe.

The solar system simulator is not just a tool for exploration and education, but also a portal to infinite possibilities. It sparks curiosity, encourages scientific inquiry, and inspires a profound appreciation for the grandeur of the cosmos. Through its blend of technology and scientific accuracy, the simulator serves as a virtual gateway to the wonders of space, inviting users to embark on a journey of discovery that transcends the boundaries of our earthly existence.

References

3D solar system orbits viewer | TheSkyLive.com. (n.d.). TheSkyLive - Your Guide to the Solar System and the Night Sky. <https://theskylive.com/3dsolarsystem>

He, C. (2020, February 7). Simulating the solar system with under 100 lines of Python code. Medium. <https://medium.com/analytics-vidhya/simulating-the-solar-system-with-under-100-lines-of-python-code-5c53b3039fc6>

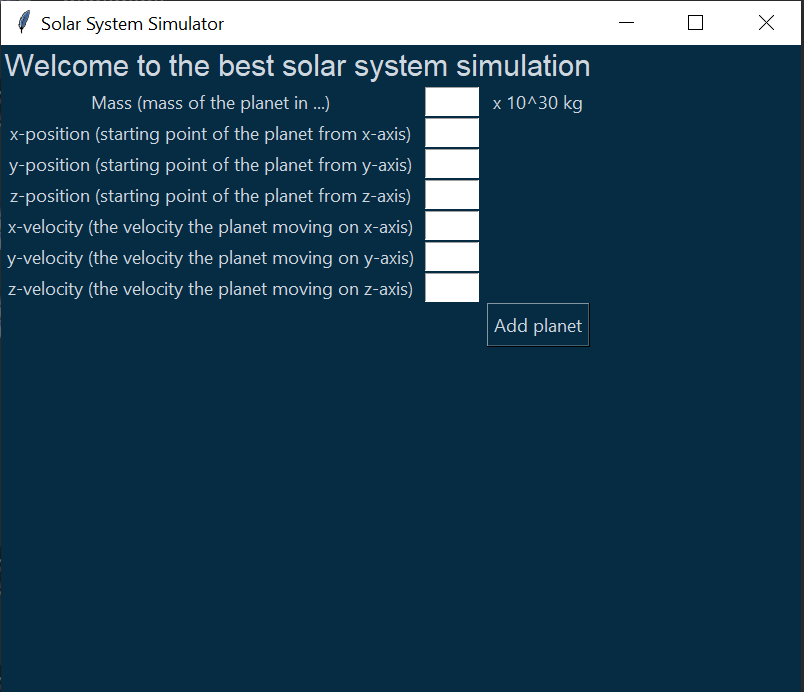
My solar system. (n.d.). PhET. <https://phet.colorado.edu/en/simulation/my-solar-system>

Zhu, A. (2022, June 10). Simulate a tiny solar system with Python. Medium. <https://towardsdatascience.com/simulate-a-tiny-solar-system-with-python-fbbb68d8207b>

Appendix

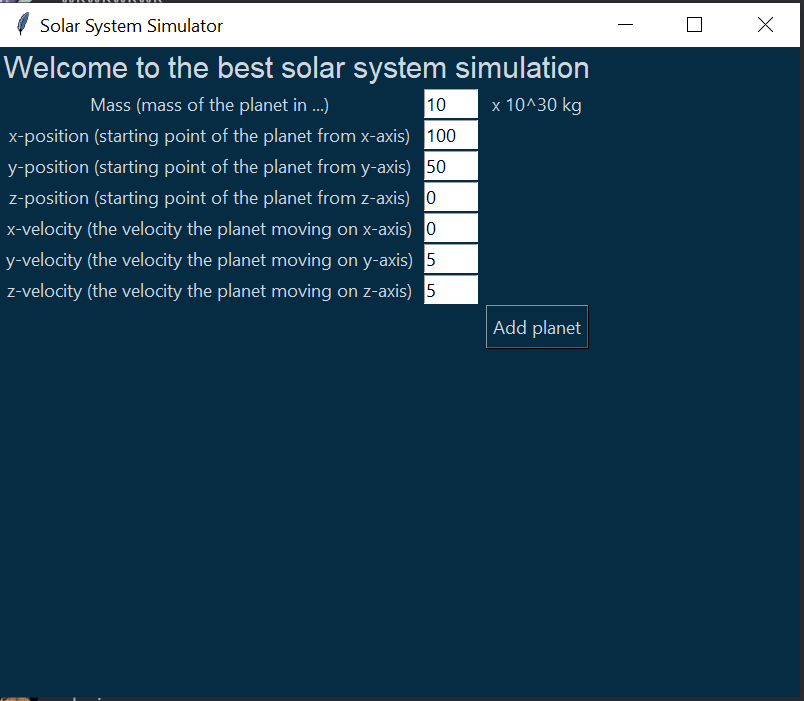
We first run the main.py file (command: python -u "main.py").

A window will pop up consisting of labels towards the variables that can be changed by the user and a textbox which allows the user to input integer or float values as shown below

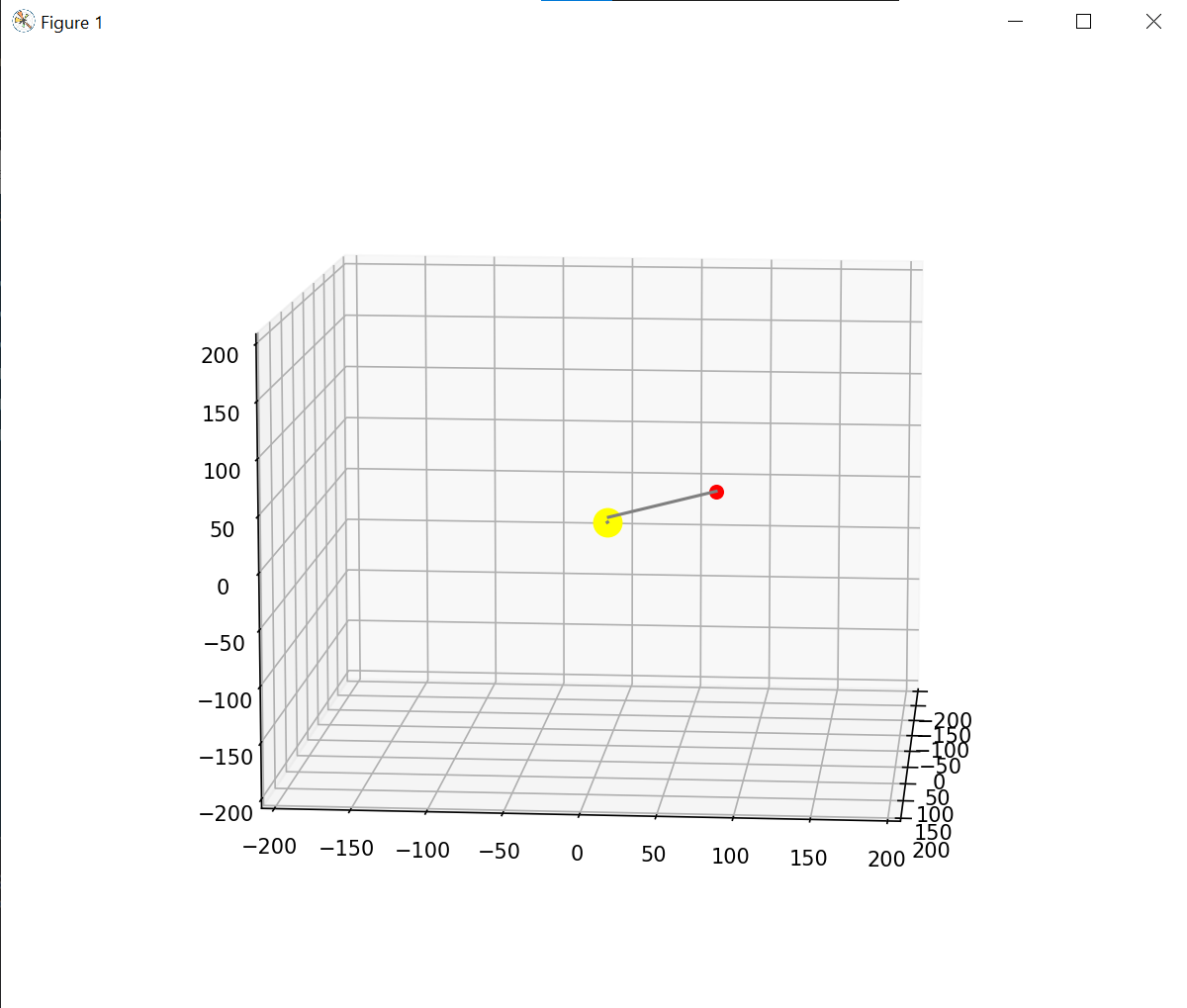


Variables that can be changed are Mass, x-position, y-position, z-position, x-velocity, y-velocity, and z-velocity.

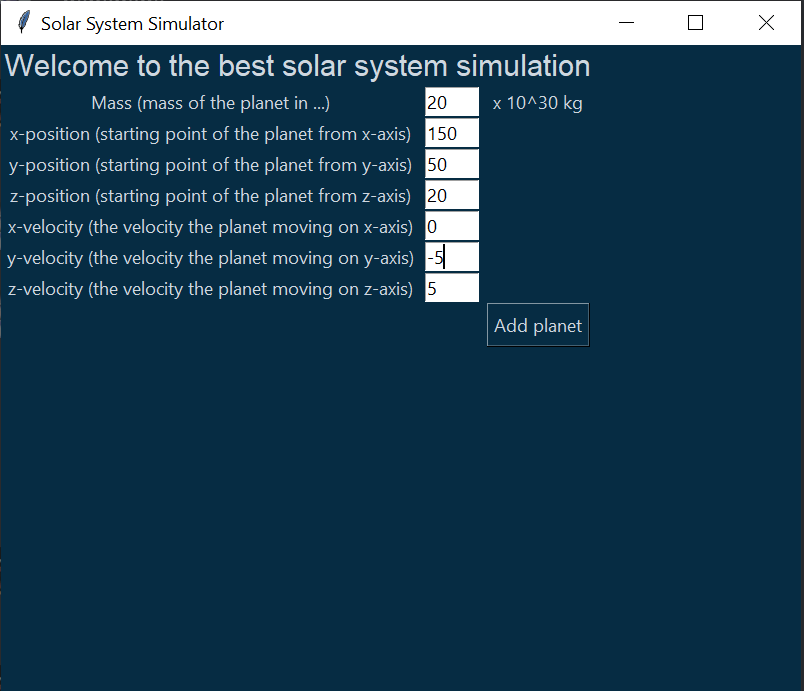
Insert the test values we want to try

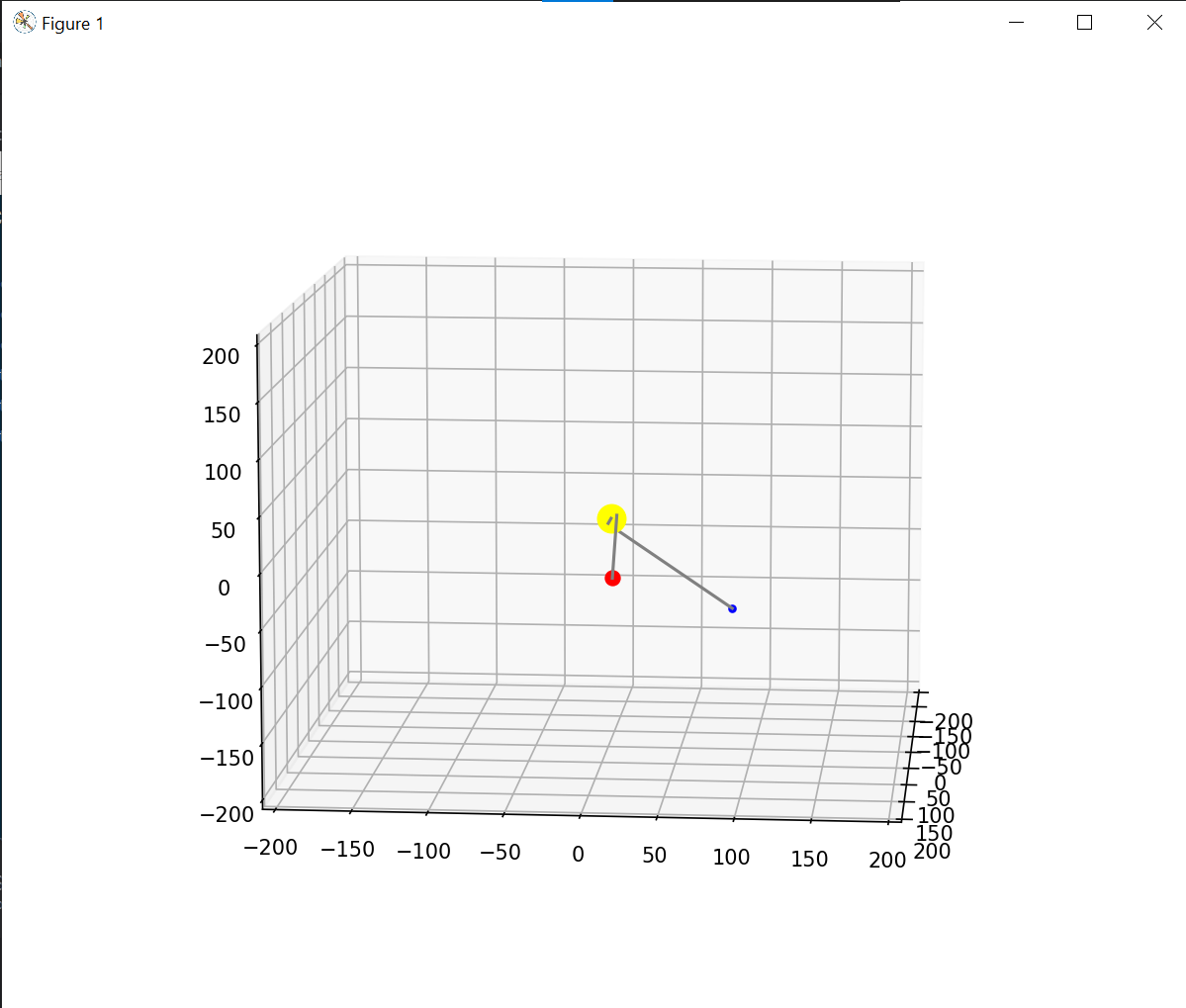


Once the user presses the “Add planet” button, a graph will pop up with a Sun and revolving planet with the position and velocity that has been taken from the user input.



To create a new planet, the user can change re-enter the variables in the input window and a planet will emerge in the graph automatically





Abdul Moiz Zahid 2602186195

Angelo Cascino 2702400400