The Class Structure:

(1) Universe Class: This class acts as the primary coordinator for the simulation, create a harmonious system. It comprises Planet object and Star object. With the addPlanet method, planets can be smoothly assimilated into the system, whereas setStar facilitates the designation of the central star. Moreover, setAsteroidBelt enables the integration of an asteroid belt into the universe. The printSystemInfo method efficiently conveys information about the system, and printAsteroidBeltLocation method show the location of the asteroid belt.

(2) Star Class: Star class inherits from the CelestialBody class and captures a star's defining features, such as luminosity, which impacts how bright it appears. The SFML graphics library visualises the star, complete with a shape object and a label denoting its name. The draw method is then called upon to showcase the star within the simulation window.

(3) Planet Class: This class extends and inherits from the CelestialBody class and serves to embody the individual planets in the simulation. Every Planet object encompasses distinct orbital traits, such as period (T), eccentricity (e), and semi-major axis (a), which are essential to computing the planet's position over time. The updatePosition method adjusts the planet's location based on the elapsed time, thus emulating its orbit around the star.

(4) AsteroidBelt Class: The AsteroidBelt class generates a belt of asteroids. It establishes the inner and outer boundaries via innerRadius and outerRadius, as well as the averageSize and asteroid quantity. The initializePositions and updatePositions methods are utilised to initialise and continuously update the asteroid positions, adding dynamism to the belt as the asteroids move within their designated area.

(5) OrbitManager Class: The OrbitManager class manages the orbital paths of the planetary objects within the simulation. It is constructed with the number of planets, initializing a vector of sf::VertexArray objects to store the visual representation of each orbit. The updateOrbits method refreshes these lines with the new positions of the planets, ensuring the orbital paths are accurately portrayed as the planets move. Through its draw method, it renders these paths onto the SFML render window, providing the trajectories that each planet follows through space.

The Main:

The main function serves as the conductor of this cosmic symphony, initialising the SFML graphics window and creating instances of the Universe, Star, Planet, AsteroidBelt and OrbitManager classes. An instance of the Star class is created to represent the central star. Subsequently, multiple Planet objects are instantiated with diverse orbital characteristics. The asteroid belt is realised by creating an instance of the AsteroidBelt class.
Once the stage is set, the simulation enters its main loop, where time becomes a variable. The positions of the planets are updated relative to the elapsed time, simulating their orbital paths. The asteroid belt's positions also update, creating a dynamic and ever-changing scene.
The drawing routine includes the drawing methods of each object. This ordered process ensures that the celestial bodies are drawn correctly relative to each other, respecting their z-order and visual hierarchy.
Here's a step-by-step breakdown of its responsibilities and workflow:

(1) Initialization: It starts by setting up the SFML video mode and creating a window where the simulation will be displayed. A clock is initiated to manage the timing and animation of the simulation.

(2) Font Loading: A font file displays text on the screen. This is important for rendering the names of celestial bodies and any other textual information in the simulation.

(3) Universe Creation: An instance of the Universe class represents the simulated universe.

(4) Star Creation: A Star object, representing the sun, is instantiated with specific attributes such as name, colour, mass, radius, and luminosity and then set as the central star of the universe.

(5) Planets Creation: The program iterates through predefined planet information, creating Planet objects for each. These planets are configured with various characteristics like colour, type, mass, radius, and orbital parameters. Each planet is added to the universe.

(6) OrbitManager Creation: An OrbitManager is instantiated with the count of planets. It's responsible for managing and visualizing the planets' orbital paths.

(7) Asteroid Belt Creation: An AsteroidBelt object represents the belt of asteroids in the simulation, adding another layer of complexity and visual interest to the universe.

(8) User Interaction for Mode Selection: Before the simulation loop begins, the user is prompted to choose between displaying text information or the animation. This choice is made through mouse interactions with text options displayed on the window.

(9) Simulation Loop: Inside the simulation loop, the program handles window events (like closing the window) and updates the state of the simulation based on the chosen mode.

(10) Text Mode: If the user selects text mode, the universe's current state, including information about the planets and asteroid belt, is displayed as text on the window.

(11) Animation Mode: If the animation mode is selected, the simulation dynamically updates and displays the positions of the planets, their orbits, and the asteroid belt based on the elapsed time. This is achieved by updating the positions of the planets, managing their orbits through the OrbitManager, and drawing the asteroid belt in motion.

(12) Rendering: The window is continuously cleared and redrawn with the updated state of the simulation, whether it be the textual information or the graphical animation of the celestial bodies and their orbits.