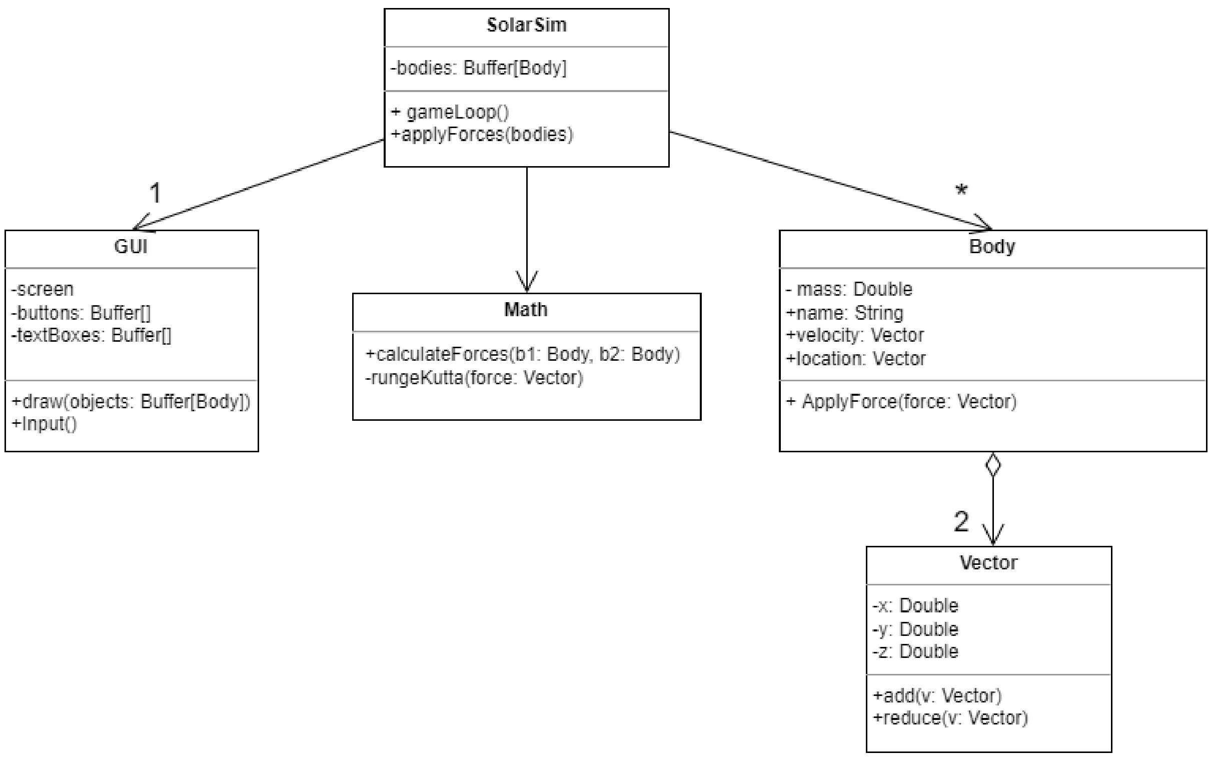
Structure plan



I will have a parent class named SolarSim that will create the GUI with some parameters. The class is going to create multiple instances of bodies and store them in a buffer. The main loop is also going to be in the SolarSim class, and the drawing and calculation of velocities will be called from the main loop. I will create my own 3-dimensional vector class that will be used as a base for storing data concerning location and velocity. The vector class will also be used as a base for doing math.

Use case description

The program will automatically load the coordinates and velocities of the planets after the user has started the application. Then the simulation should immediately start with the default time step. The simulation will happen in the Game object that will have a main loop for running the game.

The user can now interact with the program adding satellites and changing the time step. When creating new objects, the user will be asked to input coordinates, velocity, mass, and name for it. A cool feature could be the capability to add a satellite to orbit some planet. In that case, the coordinates and velocity could be inputted relative to the planet's location and velocity.

There will also be some options for adjusting the camera. These options might be for example zooming and applying different camera angles. The user interaction will happen with buttons and text boxes that belong to the GUI class and are drawn on the sides of the screen

Algorithms and data structures

The bulk of the math will be to calculate the gravitational forces and applying them to each planet. The math will be done in a separate helper object to improve readability. The algorithm that will be used for calculating the forces is going to be fourth order Runge-Kutta. There was a great tutorial linked to the project description page which I also listed down (a).

When it comes to data structures, I was thinking about storing the planets as a buffer of bodies in the Game object. Being a mutable collection, buffer allows the addition of objects to the simulation. Furthermore, each of the bodies in the simulation is going create a separate instance of vectors for both location and velocity. The use of vectors improves readability and shortens the code for not needing to always specify x, y and z components separately.

Timetable

My timetable starts 28.02.2022. Here it is week by week.

1. Creating GUI.
2. Creating GUI.
3. Creating Body and Vector classes
4. Creation of the SolarSim object
5. Creation of the SolarSim object
6. Implementing math and Runge-Kutta
7. Implementing math and Runge-Kutta

8. Debugging and implementing additional features like camera control.
9. Debugging and implementing additional features.

Test design/plan

There should be a way to test the GUI independently. If the need for individual GUI testing arises, a test object can be created. In that case some of the functions that call other classes will be disabled or some dummy objects will have to be created. Additionally, the functions in the Math object should not depend on any other class or object and as such they should be able to be tested independent from the rest of the application.

In case of faulty input, the program should output an error message on the screen or in the console. The error message should describe to the user how to correctly form the input data.

Links

a. https://gafferongames.com/post/integration_basics/