

# Contents

Analysis	1
Introduction	1
Research	1
Modelling	1
Prototype	3
Final Objectives	3
Design	4
Mechanics	4
Mathematical Structures	4
Bodies	5
PlanetarySystem	6
Graphics	7
SystemView	7
Camera	8
UI	9
Save/Load	10
Input	10
Pausing	11
Control Flowchart	11
Class Diagram	12
Tests	13
Vector/Matrix	13
Body/OrbitalElements	13
PlanetarySystem	14
System Tests	14
Test Screenshot/Videos	15
Evaluation	15
Interview	15
Objective Analysis	16
Conclusion	17
Appendices	>17

# Analysis

## Introduction

This project is in the field of astrodynamics, the study of the mechanics of orbits in solar systems. This subject is very difficult to give a visual intuition of – the textbooks are full of difficult equations, and few give a real demonstration of how the different variables affect orbits.

This project aims to provide a useful simulation where users can input and edit conditions and see a visual representation of various orbits. This can be used for students trying to learn more about astrodynamics, and teachers trying to give a visual demonstration.

## Research

In conversations with potential users I found that the most important aspect of this program is the modelling accuracy, but also particularly the graphical representation during the simulation – since it is modelling 3D space, it should be possible in some way to navigate the space to fully picture the results of the simulation.

There are two existing programs which provide a similar service to what is outlined in the Introduction – Kerbal Space Program, and Universe Sandbox. Kerbal Space Program primarily aims to provide a tool for creating your own space program – building rockets and space stations, managing resources, and so on, but the astrodynamics of the planetary orbits and spacecraft orbit is based on real physics – the patched-conic approximation (See below). It is also limited in that the planets follow fixed paths and cannot be perturbed by other bodies. Universe Sandbox provides a full gravitational simulation based on small timesteps of Newton's Law of Gravitation, but it is relatively expensive (~£20) which does not provide for teachers in underfunded schools who want to show a simulation for maybe only a lesson or two each year.

## Modelling

There are two main methods of approximating gravitational motion, which were briefly mentioned above.

### **The patched conic approximation:**

Since in most planetary systems there is one body that is much heavier than the others, (e.g. the sun), the simplest approximation would be to ignore all interactions between pairs of lighter bodies and model only interactions between each body and the most massive body. This would be a conic approximation, since in this instance all paths would be conic sections (or degenerate conic sections, such as a fixed point or a straight line)

The patched conic approximation is slightly more complex than this, in that for each body it defines a “Sphere of Influence”, which is the sphere within which this body is the most gravitationally significant body. Then interactions for a less massive body inside this sphere of influence can be modelled as a conic section with the gravitationally significant body as a focus.

For example, for a spacecraft orbiting the moon, gravitational attraction from Earth and the sun is very small, so its orbit can be modelled as an ellipse with the moon as a focus. If this spacecraft were to accelerate away from the moon, it would soon exit the “sphere of influence” of the moon and it could be modelled as following the path of a conic section with Earth as the focus.

## Newtonian timestep:

The instantaneous acceleration vector on each body can be determined precisely as

$$\underline{a}_b = \sum_{i=2}^{i=n} (\underline{r}_i - \underline{r}_b) \frac{G m_i}{|\underline{r}_i - \underline{r}_b|^3}$$
, where  $n$  is the number of bodies in the system,  $\underline{r}_b$  is the position vector of the body,  $\underline{r}_i$  is the position vector of another body,  $m_i$  is the mass of the other body, and  $G$  is the gravitational constant. Since  $\underline{r}_i = \underline{r}_{i0} + \int_{t=0}^{t=t_{now}} \underline{a}_i dt$ , it is quite quickly becoming clear that the equation of the position of a body at a certain time may be unsolvable (and indeed it is!).

The Newtonian timestep approximation attempts to approximate a solution to this double integral via numerical methods:

$$\underline{r}_{b,n+1} = \underline{r}_{b,n} + \int_{t=t_n}^{t=t_{n+1}} \underline{a}_b dt \approx \underline{r}_{b,n} + \underline{v}_{b,n}(t_{n+1} - t_n) + \frac{1}{2} \underline{a}_{b,n}(t_{n+1} - t_n)^2$$
, where

$$\underline{v}_{b,n+1} = \int_{t=t_n}^{t=t_{n+1}} \underline{a}_b dt \approx \underline{v}_{b,n} + \underline{a}_{b,n}(t_{n+1} - t_n)$$
, where  $\underline{a}_{b,n}$ ,  $\underline{v}_{b,n}$ , and  $\underline{r}_{b,n}$  are the acceleration, velocity, and position vectors of body  $b$  at time  $t_n$ . In this case, we must define a “timestep”  $t_{n+1} - t_n$ . The lower this timestep is, the more accurate the approximation will be, but the more calculations (and hence more real-world time) required for the same amount of in-simulation time.

Here is a comparison

Newtonian Timestep	Patched-Conic Approximation
Can model complex interactions accurately (e.g binary star systems)	More accurate for most simple systems (eg. our solar system)
Without enough processing power, it will either run slowly or inaccurately	Can run arbitrarily fast
	Less processor intensive

If the mechanics can be done directly from Newton’s law of gravitation and remain accurate and fast, then it will be a much better option than the patched conic approximation, since it gives much greater flexibility in the types of situations which can be modelled. To test this, I built an early prototype of the mechanics system to find out. The test was as follows: Approximate initial conditions of an Earth-Sun system by hand, enter the conditions into the mechanics algorithm, and test running time against the change in specific energy, since the change in specific energy is the

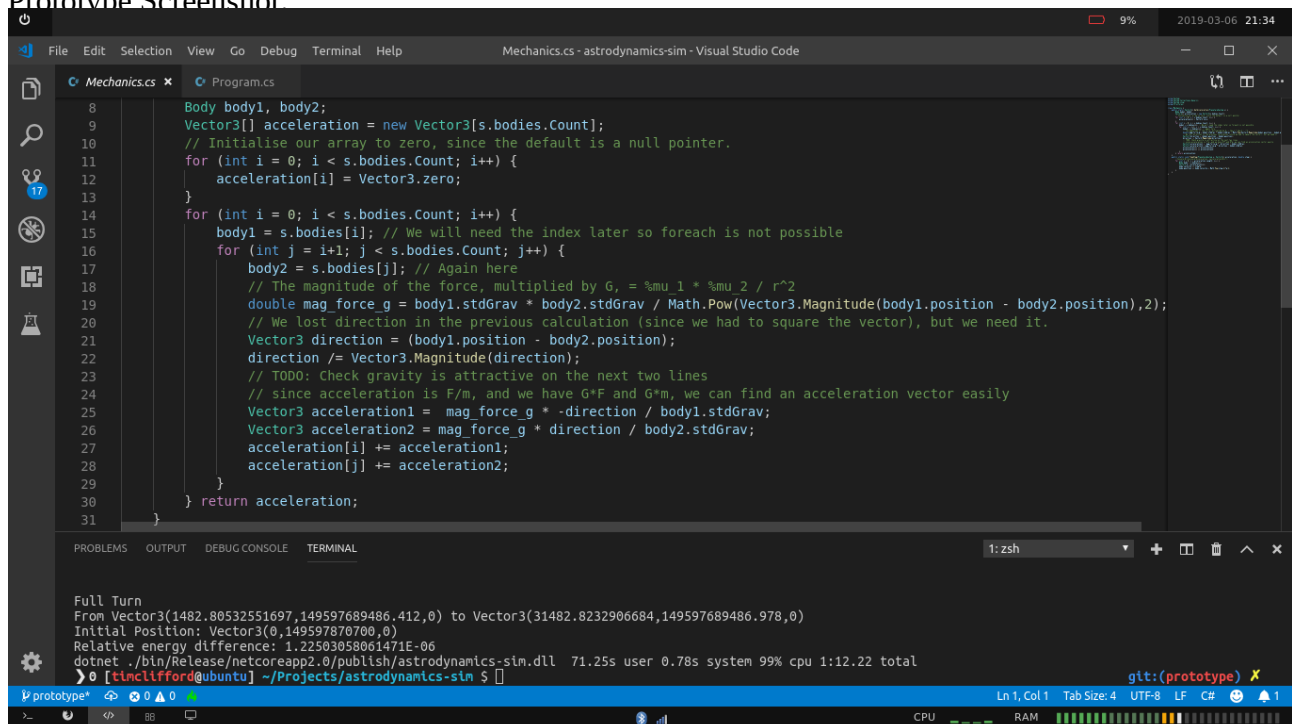
## Astrodynamics Simulation

most important consequence of an inaccurate algorithm – the planets will begin to spiral away from the object they should be orbiting rather than remaining bound.

The result was that at a real-life orbit period of a few seconds the change in energy was +0.1%, and at an orbit period of a minute, it dropped to +0.001%. This was on a very low-powered machine, so this amount of error is more than could be expected on a modern computer.

In a conversation with a potential user, we agreed that this would be an acceptable amount of error accumulation, since it will not be noticeable to the human eye, and since the program would be primarily used as an educational tool rather than professionally.

### Prototype Screenshot:



```
8 Body body1, body2;
9 Vector3[] acceleration = new Vector3[s.bodies.Count];
10 // Initialise our array to zero, since the default is a null pointer.
11 for (int i = 0; i < s.bodies.Count; i++) {
12     acceleration[i] = Vector3.zero;
13 }
14 for (int i = 0; i < s.bodies.Count; i++) {
15     body1 = s.bodies[i]; // We will need the index later so foreach is not possible
16     for (int j = i+1; j < s.bodies.Count; j++) {
17         body2 = s.bodies[j]; // Again here
18         // The magnitude of the force, multiplied by G, = %mu_1 * %mu_2 / r^2
19         double mag_force_g = body1.stdGrav * body2.stdGrav / Math.Pow(Vector3.Magnitude(body1.position - body2.position),2);
20         // We lost direction in the previous calculation (since we had to square the vector), but we need it.
21         Vector3 direction = (body1.position - body2.position);
22         direction /= Vector3.Magnitude(direction);
23         // TODO: Check gravity is attractive on the next two lines
24         // since acceleration is F/m, and we have G*F and G*m, we can find an acceleration vector easily
25         Vector3 acceleration1 = mag_force_g * -direction / body1.stdGrav;
26         Vector3 acceleration2 = mag_force_g * direction / body2.stdGrav;
27         acceleration[i] += acceleration1;
28         acceleration[j] += acceleration2;
29     }
30 } return acceleration;
31 }
```

Full Turn  
From Vector3(1482.80532551697, 149597689486.412, 0) to Vector3(31482.8232906684, 149597689486.978, 0)  
Initial Position: Vector3(0, 149597870700, 0)  
Relative energy difference: 1.22503058061471E-06  
dotnet ./bin/Release/netcoreapp2.0/publish/astrodynamics-sim.dll 71.25s user 0.78s system 99% cpu 1:12.22 total  
[tinclifford@ubuntu] ~/Projects/astrodynamics-sim \$

We also agreed that for ease of use it would be best to describe the bodies as following conic orbits, and then convert into a pure position and velocity before starting the simulation.

Since users will want to be able to tweak different variables a lot, we agreed that it would be very good also, if possible, to provide a tool to save configurations to the hard drive and load them again later.

## Final Objectives:

1. Accurately model simple Keplerian orbits – circular, elliptical, parabolic, and hyperbolic
2. Provide a tool to add, remove, and edit bodies with classical orbital elements, and provide an explanation:

## Astrodynamics Simulation

Name	Description	Symbol	Range
Semi-latus rectum	the distance between two bodies at right angles to the “periapsis” (minimum point)	$\rho$	$[0, +\infty)$
Eccentricity	A measure of the shape of the orbit	$e$	$[0, +\infty)$
Inclination	the angle between the orbital plane and the reference plane	$i$	$[0, 180^\circ]$
Longitude of the ascending node	the angle from the reference direction anticlockwise to the point where the orbiting body rises above the reference plane	$\Omega$	$[0, 360^\circ)$
Argument of periapsis	the angle from the ascending node anticlockwise to the periapsis.	$\omega$	$[0, 360^\circ)$
True anomaly	The angle from the periapsis anticlockwise to the current position of the body.	$v$	$[0, 360^\circ)$

3. Simulate motion in non-classical systems – e.g a rogue planet entering the solar system
4. Render the 3D system into 2D convincingly.
5. Be able to place one or more cameras anywhere in the system and provide controls for zoom/rotation/etc.
6. Be able to pause the simulation and modify variables at any point.
7. Provide a capability to save and load systems to/from the hard drive.

## Design

This is split into three main areas: Mechanics, Graphics, and UI, which have been approached separately. Mechanics deals with calculations of how the bodies are affected with time, Graphics deals with the 3D projection of the simulation to the user while it is running, and UI deals with the process of adding, removing, and editing bodies.

## Mechanics

### Mathematical Structures:

We will need some basic mathematical structures to make 3D calculations easier: the 3-Vector, which stores 3 real numbers, and can be used for storing positions, velocities, and so on, and the 3x3 Matrix, which makes operations on 3-Vectors like rotation easier. These will also be useful for the Graphics process. These data types should be able to perform simple mathematical functions implicitly and provide static functions for more complex operations such as the dot and cross product.

Since the double type will be used for the simplest part of these structures, and since after repeated computation some small errors will accumulate, the equality of these structures will check that each component value is within a small value of each other rather than exactly the same. By experimentation, a value of  $10^{-10}$  was deemed suitable.

The implementation of these structures can be seen in Appendix 1.

## Bodies:

There must be a data structure which describes a body in the system. This will be stored in main memory as the program is running, but also must be able to be saved to disk.

Since we chose to use a Newtonian timestep approximation, the three variables required for mechanics are:

StdGrav – the standard gravitational parameter of the body.

Position – the current position of the body (as a vector)

Velocity – the current velocity of the body (as a vector)

For identification of bodies and initial orbit calculations the following two are also required:

Name – the name of the Body

Parent – the body this body is orbiting

And for the Graphics process:

Color – the color of the body

This class should be able to be constructed from the six orbital elements, so an “OrbitalElements” class is required, storing these elements.

The process of converting an orbit determined by the six orbital elements into an initial position and velocity is as follows:

```
// First, find the position and velocity in “perifocal coordinates”, which means that
// the orbital plane is the X-Y plane, and the periapsis is along the x axis
```

$$\underline{r}_p \leftarrow \frac{\rho}{1+e\cos v} \begin{pmatrix} \cos v \\ \sin v \\ 0 \end{pmatrix} ; \quad \underline{v}_p \leftarrow \sqrt{\frac{\mu}{\rho}} \begin{pmatrix} -\sin v \\ \cos v + e \\ 0 \end{pmatrix}$$

```
// Now we must convert these into common IJK coordinates. This is in fact several
// rotations, from which a matrix was calculated by hand.
```

$$M \leftarrow \begin{pmatrix} \cos \Omega \cos \omega - \sin \Omega \sin \omega \cos i & -\cos \Omega \sin \omega - \sin \Omega \cos \omega \cos i & \sin \Omega \sin i \\ \sin \Omega \cos \omega + \cos \Omega \sin \omega \cos i & -\sin \Omega \sin \omega + \cos \Omega \cos \omega \cos i & -\cos \Omega \sin i \\ \sin \omega \sin i & \cos \omega \sin i & \cos i \end{pmatrix}$$

$$\underline{r} \leftarrow M \underline{r}_p ; \quad \underline{v} \leftarrow M \underline{v}_p$$

The implementation of the Body class can be found in Appendix 2

Since we also want to be able to pause the simulation and view/edit variables, we need also to be able to run this algorithm in reverse. This is more complex, since it will contain arcos(), so we must ensure that our angles are in the correct quadrant.

We will first calculate the “Fundamental Vectors”, which are a sort of interim state between position-velocity and orbital elements. This will be implemented as its own class, with the calculation in the constructor. These vectors are as follows:

## Astrodynamics Simulation

// Specific Angular Momentum: This is the angular momentum of the body around its parent, divided by the mass of the parent. It can be shown to be constant in any conic orbit.

$$\underline{h} \leftarrow \underline{r} \times \underline{v}$$

// Eccentricity: a vector in the direction of the periapsis, with magnitude of the eccentricity of the orbit

$$\underline{e} \leftarrow \underline{r} \left( \frac{v^2}{\mu} - \frac{1}{r} \right) - (\underline{r} \cdot \underline{v}) \times \underline{v}$$

// Node: The direction of the node where the orbital plane intersects the I-J plane

$$\underline{n} \leftarrow \underline{h} \times \underline{K} \quad // \quad \underline{K} := (0, 0, 1)$$

From these vectors we can then calculate the orbital elements:

$$e \leftarrow |\underline{e}| \quad ; \quad \rho \leftarrow \frac{h^2}{\mu} \quad ; \quad i \leftarrow \arccos(\hat{\underline{h}} \cdot \underline{z})$$

$$\cos \Omega \leftarrow \hat{\underline{n}} \cdot \underline{x}$$

IF ( $\hat{\underline{n}} \cdot \underline{y} \geq 0$ ) THEN  $\Omega \leftarrow \arccos(\cos \Omega)$

ELSE  $\Omega \leftarrow 2\pi - \arccos(\cos \Omega)$

$$\cos \omega \leftarrow \hat{\underline{e}} \cdot \hat{\underline{e}}$$

IF ( $\hat{\underline{e}} \cdot \underline{z} \geq 0$ ) THEN  $\omega \leftarrow \arccos(\cos \omega)$

ELSE  $\omega \leftarrow 2\pi - \arccos(\cos \omega)$

$$\cos v \leftarrow \hat{\underline{e}} \cdot \hat{\underline{r}}$$

IF ( $|\underline{r}| \cdot |\underline{v}| \geq 0$ ) THEN  $v \leftarrow \arccos(\cos v)$

ELSE  $v \leftarrow 2\pi - \arccos(\cos v)$

In implementation this algorithm must be modified, since double precision floating point error leads to nonsense values for  $v$  at small values of  $e$  and  $\Omega$ . It is assumed that if  $e$  is small, the periapsis is at  $\Omega$ , and if  $\Omega$  is also small, it is the I vector.

The implementation of this algorithm can be seen in Appendix 3

### PlanetarySystem:

We also need a class to store the planetary system as a whole. This should inherit from the standard IEnumerator interface, which allows the bodies which are stored in it to be iterated through, indexed, added to, and removed from. It will store the bodies as a protected list. This simplifies access to the system from other objects.

Bodies – a list of the bodies in the system

Centers – the index of each body in the system which can be focused on when the system is drawn.

The planetary system class will also handle all of the mechanics of the system. As it was decided to use Newton's laws, the instantaneous acceleration on each body must be calculated first.

```
foreach body A in bodies:
    foreach body B in bodies:
        accelerationA ← accelerationA + b.stdGrav/(distance between them squared)
        accelerationB ← accelerationB + a.stdGrav/(distance between them squared)
```

This runs in  $O(n^2)$  time, which is acceptable. This calculation should be in a function called GetAcceleration().

## Astrodynamics Simulation

The PlanetarySystem class will include a method TimeStep(step), which takes the instantaneous accelerations and assumes they are constant for a small time “step” in seconds, and modifies the bodies.

```
foreach body B in bodies:
    B.position += B.velocity*timestep + (timestep^2/2)*accelerationB
```

The calculations should be encapsulated within the class and run asynchronously to the other processes. Public Start(), StartAsync(), and Stop() functions should be provided. These rely on a flag in the class which says whether the program is running.

The implementation of the PlanetarySystem class can be seen in Appendix 4

## Graphics:

Gtk# was chosen for this project as a graphics library since it is used professionally, has good support, and runs well on Linux with the Mono runtime.

The graphics processor should iterate through the planetary system class asynchronously from the mechanics, and, in order of distance to the camera, draw each object (and it's associated trail), passing it through a camera transformation to project it onto the screen.

Required classes:

Camera – contains an orientation and distance to the origin, as well as transformation function  
SystemView – overloads an existing GTK class and defines what happens when the screen is drawn.

Variables:

- active planetary system
- list of previous positions of bodies (to draw path)

Variables which are modified by Input class:

- planetary radius multiplier
- trail length
- active camera

Constants:

- line width (as a fraction of planetary size)

Pseudocode:

```
Fill screen with black
Set (0,0) as centre of screen
Scale coordinates to screen
order ← order of bodies according to distance from camera (closest first)
foreach Body in system according to order:
    DrawCircle(radius Body.radius*radius multiplier, position: camera transformation
of Body.position of color Body.color
    lastPath ← first point in paths[Body]
    foreach (point in paths[Body]):
        DrawLine(camera transform of lastPath to camera transform of point, width:
Body.radius*radius multiplier*line width, color Body.color)
    if len(paths) > maximum trail length:
        paths ← last (maximum trail length) elements of paths
```



## Astrodynamics Simulation

GTK is event based, which means that ordinarily the screen would only be refreshed if an event happens, eg the player clicks on something. Therefore we must create our own method to manually trigger the draw function quickly so that the simulation can be seen. The starting and stopping is achieved via a flag – it is very important to terminate all processes when the program is stopped, since we have many asynchronous processes which could theoretically carry on even if the main thread is stopped.

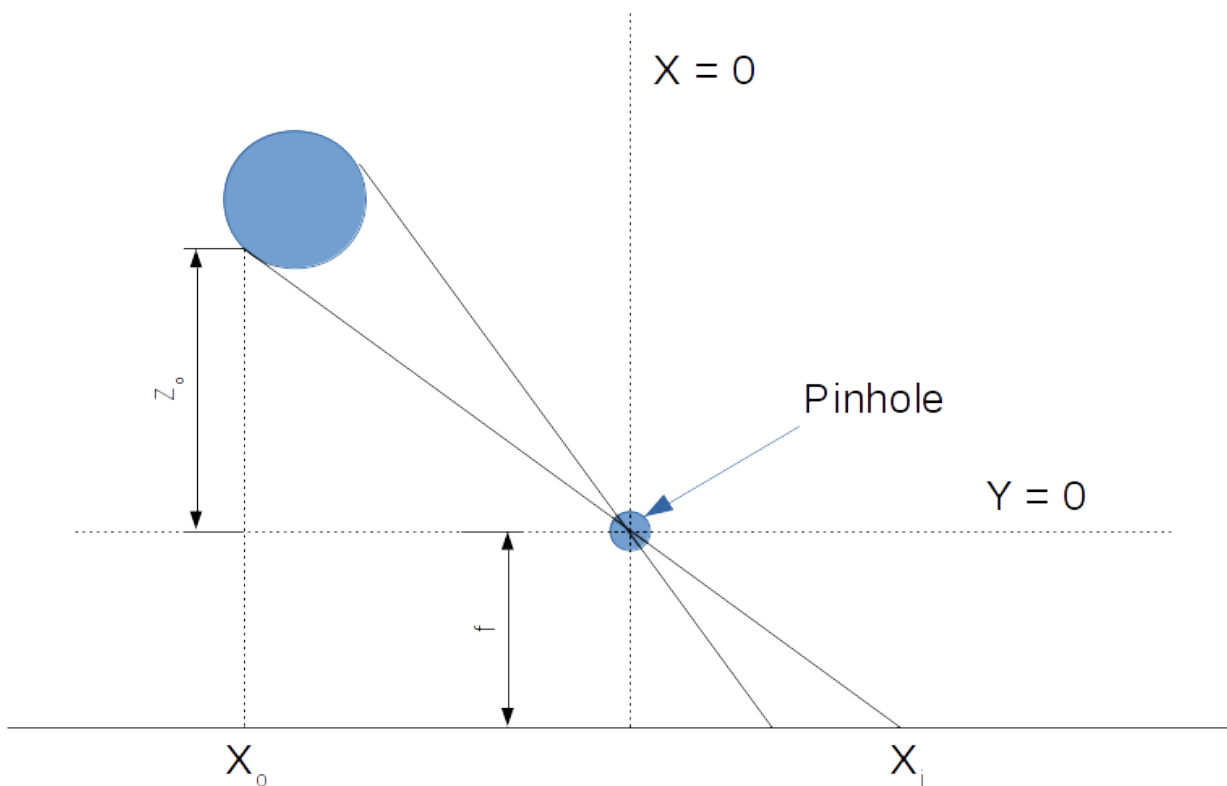
The implementation of the SystemView class can be seen in Appendix 5

### Camera Transformation:

Since we want to be able to focus on a certain body and still move the camera around, a camera is modelled as a point on a sphere which looks directly at the origin. This requires two variables: an angle (relative to the IJK vector system), and a distance from the origin.

I chose to model the camera as a pinhole camera, because the implementation is relatively simple and it provides a projection which looks very accurate. This is where light to the camera passes through a “pinhole” and lands on a screen behind it. The location of each point on the screen is it’s projection onto our 2D screen.

Diagram:



## Astrodynamics Simulation

By similar triangles,  $X_i = -X_o \frac{f}{Z_o}$ , but since we do not want to flip directions in our simulation, we will use  $X_i = X_o \frac{f}{Z_o}$ . This formula is exactly the same for a 3<sup>rd</sup> dimension:  $Y_i = Y_o \frac{f}{Z_o}$ .

By the same argument the radius of a sphere which is projected in this way is  $r_i = r_o \frac{f}{Z_o}$  in the case that  $r_o \ll d$ , so this will be the radius of our objects.

To apply this algorithm we must first transform the coordinates such that the position of the camera is (0,0,0) and the vector (0,0,1) points towards the “origin”. This can be achieved simply using our Matrix3 class.

Pseudocode:

```
SUB CameraTransform(position, camera, origin)
# position is a Vector3 object, camera is a Camera object
# using Tait-Bryan angles
cameraLocalPosition ← Matrix3.IntrinsicZYXRotation(camera.angle)
    *Vector3(0,0,-camera.focalLength)
localPosition ← position - cameraLocalPosition - origin
localPositionRotated ← Matrix3.ExtrinsicZYXRotation(camera.angle) *localPosition
OUTPUT (camera.focalLength/localPositionRotated.Z)*localPositionRotated
# the Z value of the output will not be used,
# so it does not matter that it is transformed like X and Y
```

The camera transformation can be made “orthographic” (i.e. distances are not distorted), by increasing the focal length to a very large value. This will be an option to the user via a key command.

The implementation of the camera can be seen in Appendix 6

## UI:

The UI must have functionality to set variables such as the mechanics timestep, trail length, and planetary radii multipliers, as well as add and remove bodies, set body names, set body variables, save and load the current state.

Since most of the data in this menu will be specific to a body, it makes sense to define a separate class that describes a box containing the body variables/sliders etc.

The required properties:

name: text box  
parent: drop down menu  
mass: slider  
radius: slider  
semilatusrectum: slider  
eccentricity: slider  
inclination: slider

## Astrodynamics Simulation

ascendingNodeLongitude: slider  
periapsisArgument: slider  
trueAnomaly: slider  
focusable: check button (allows the body to be set as the centre of the camera projection)  
delete: button

This class should also reference the body it describes, and be able to set the body variables based on the sliders, or set the sliders based on the body. It should also be able to send a message to the parent menu class to remove it if the delete button is pressed.

It is linked to the body via aggregation. This decouples the UI process from the Mechanics which happen later – when the UI Process terminates itself on the “Done” button click, the bodies it created will not also be destroyed.

The UI Implementation can be seen in Appendix 7

Since the variables are so difficult to understand, we should also have some sort of help function. The best way to do this would be to link to a web page, since this allows for much better formatting than a textbox inside a program. Since web design is not the focus of this project, I decided to create a markdown file containing the help and convert it to HTML via 3<sup>rd</sup> party software.

Final Markdown and rendered HTML is in Appendix 8

### **Save/Load:**

I chose XML as the file format to save into, since it is human readable but still offers good flexibility as to the types that are saved. I decided a custom save file class should be created, acting as a wrapper for the data that must be saved.

Required data:

- Mechanics Timestep
- Planetary Radii multiplier'
- Orbit Trail Length
- Body variables
- which bodies can be focused on

Several Example systems are provided as XML files. These are outlined in the help file, and provide a tool for new users to understand the capabilities of the system without having to create their own system. An example (Our solar system) is shown in Appendix 9

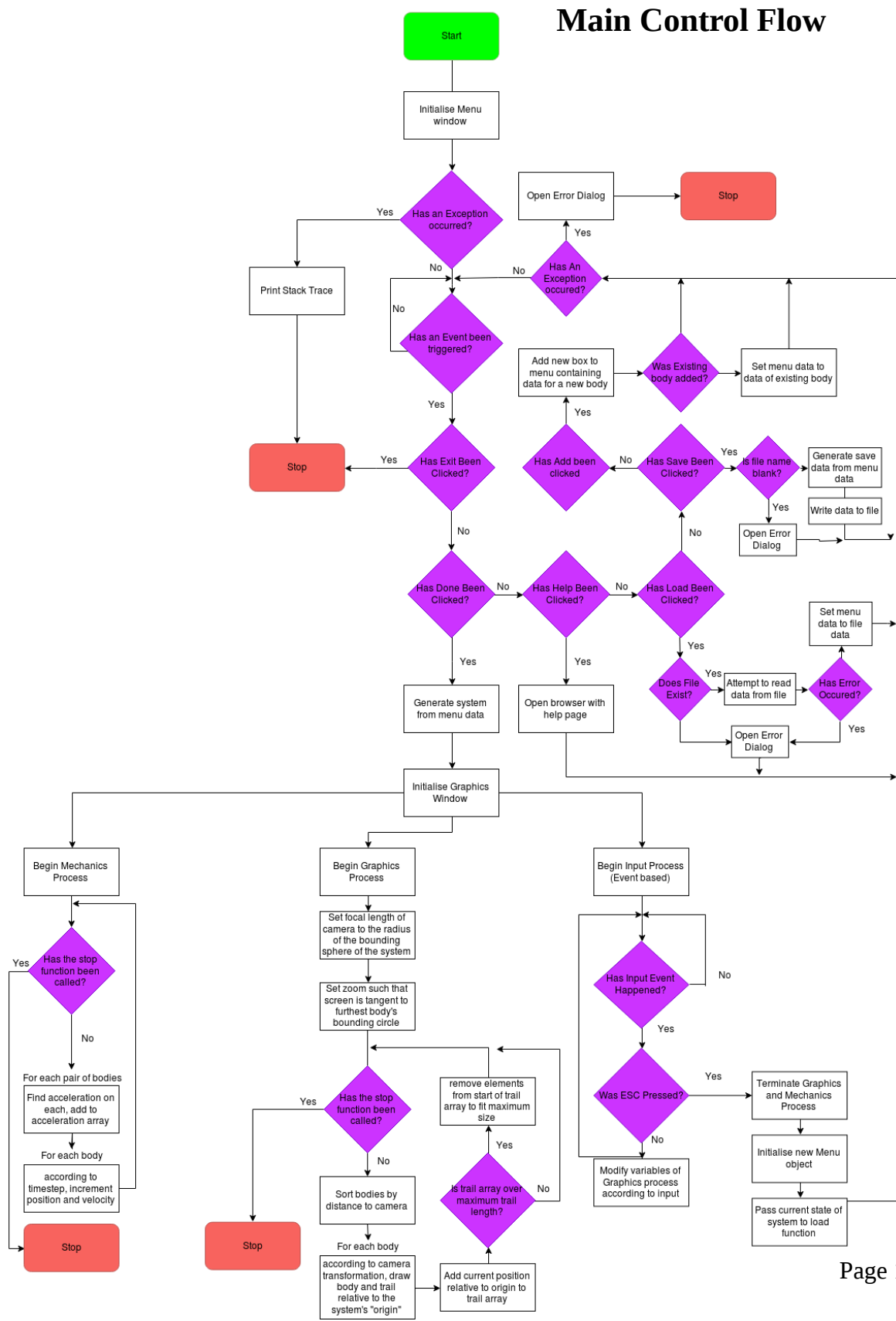
### **Input**

In simulation input will be achieved via Event-based triggering. EventHandlers are defined for each type of input, which have access to the active PlanetarySystem and SystemView objects via the Program class, which starts the program and stores the active objects as public properties.

The implementation can be seen in Appendix 10

## Pausing

I decided that the best way to allow users to stop and edit the variables of the system would be to take them back to the original setup UI screen. To take advantage of the already defined functions I decided to generate a new SaveData object, but instead of writing it to a file, to create a new UI object and run the Load command, passing this SaveData as an object. Therefore the user can stop the simulation whenever he/she wants and modify the orbit variables. This control flow can be seen in the main control flowchart below.





## Tests

Tests were included as a part of the design of data structures. These are outlined below.

### Vector/Matrix

Class	Test	Type	Expected	Pass/Fail
Vector3	$(2,3,6) + \text{zero} = \text{zero} + (2,3,6) = (2,3,6)$	Normal	True	Pass
Vector3	$5 * (2,3,6) / 5$	Normal	(2,3,6)	Pass
Vector3	$(2,3,6) + (2,3,6) = 2 * (2,3,6)$	Normal	True	Pass
Vector3	$-a = 0 - a$	Normal	True	Pass
Vector3	Magnitude(2,3,6)	Normal	7	Pass
Vector3	$(1,2,0) \cdot (-2,1,0)$	Normal	0	Pass
Vector3	$(2,3,6) \cdot (2,3,6)$	Normal	49	Pass
Vector3	$(3,-3,1) \times (4,9,2)$	Normal	(-15,-2,39)	Pass
Vector3	Unit(2,3,6)	Normal	(0.285...,0.428...,0.857...)	Pass
Vector3	Unit(0,0,0)	Erroneous	DivideByZeroException	Pass
Vector3	PolarToCartesian(CartesianToPolar((2,3,6)))	Normal	(2,3,6)	Pass
Vector3	Null = null	Erroneous	True	Pass
Vector3	Null = (2,3,6)	Erroneous	False	Pass
Matrix3	$I * I = I$	Normal	True	Pass
Matrix3	$I * [1,3,1],[0,4,1],[2,-1,0] = [1,3,1],[0,4,1],[2,-1,0] * I = [1,3,1],[0,4,1],[2,-1,0]$	Normal	True	Pass
Matrix3	$5 * ([1,3,1],[0,4,1],[2,-1,0]) / 5$	Normal	([1,3,1],[0,4,1],[2,-1,0])	Pass
Matrix3	$[1,3,1],[0,4,1],[2,-1,0] + [1,3,1],[0,4,1],[2,-1,0] = 2 * [1,3,1],[0,4,1],[2,-1,0]$	Normal	True	Pass
Matrix3	Inverse([1,3,1],[0,4,1],[2,-1,0])	Normal	([-1,1,1],[-2,2,1],[8,-7,-4])	Pass
Matrix3	Inverse(Inverse([1,3,1],[0,4,1],[2,-1,0]))	Normal	([1,3,1],[0,4,1],[2,-1,0])	Pass
Matrix3	Inverse([0,0,0],[0,0,0],[0,0,0])	Erroneous	DivideByZeroException	Pass
Matrix3	$([1,3,1],[0,4,1],[2,-1,0]) * ([1,3,1],[0,4,1],[2,-1,0])$	Normal	([3,14,4],[2,15,4],[2,2,1])	Pass

### Body/OrbitalElements Tests

Test	Type	Expected	Pass/Fail
Identical bodys created orbiting static and moving bodies	Normal	Created bodies' position and velocity differ by that of the moving body	Pass
OrbitalElements(Body(all possible values of variables, in increments of 0.2))	Normal	Equal to the original variables, and all circular orbits of the same radius have the same velocity	Pass
OrbitalElements(angles greater than 2pi)	Erroneous	Angles are implicitly converted to be within bounds	Pass

## PlanetarySystem tests

Test	Type	Expected	Pass/Fail
Construct system from list of bodies	Normal	System contains bodies	Pass
Add body to system	Normal	System contains body	Pass
Barycenter of two bodies on line	Normal	Returns position on line in ratio of gravities	Pass
Barycenter of bodies on vertices of equilateral triangle	Normal	Returns center of triangle	Pass
Specific Energy drift while simulating (tested as prototype)	Normal	Not Significant	Pass

The implementation of these tests is in Appendix 11

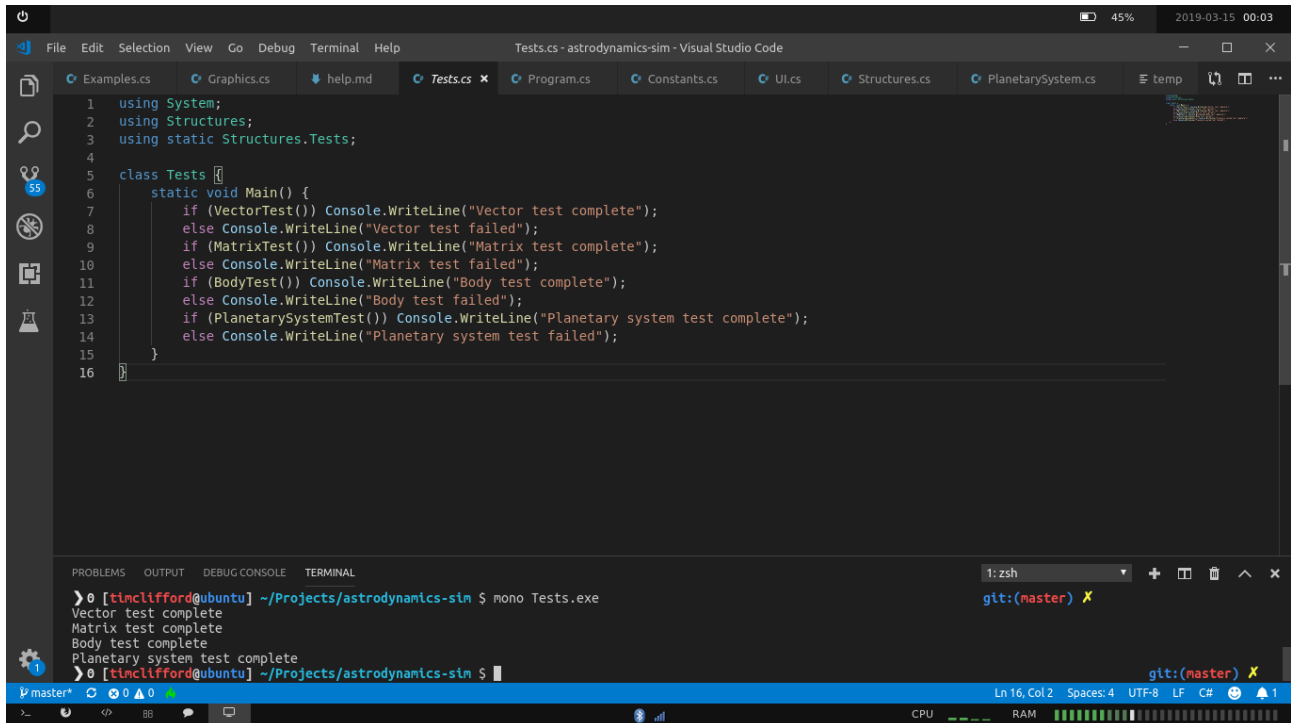
## System Tests

The Graphics and UI were tested qualitatively as part of the entire system.

Test	ID	Type	Expected	Pass/Fail
UI provides functional buttons/sliders for all options	1	Normal	Buttons work as labelled	Pass
Invalid filename is entered and load button is pressed	2	Erroneous	UI shows error message	Pass
File is saved, program is exited, and file is loaded again	3	Normal	Options are exactly the same	Pass
ESC is pressed during simulation	4	Normal	Simulation exits and UI menu is loaded with current system parameters	Pass
F is pressed during simulation	5	Normal	Camera focuses on next body	Pass
C is pressed during simulation	6	Normal	Camera switches to stereoscopic mode	Pass
Scroll wheel and mouse are used during simulation	7	Normal	Camera zooms and moves	Pass
Arrow keys and PgUp/PgDown are pressed	8	Normal	Graphics variables are changed	Pass
System is started with rogue planet in hyperbolic orbit	9	Normal	Believable non keplerian motion, orbital parameters change.	Pass

## Test Screenshots/Videos:

Data structures test screenshot:



The screenshot shows the Visual Studio Code editor with a C# file named `Tests.cs` open. The code defines a `Tests` class with a `Main` method that calls several test functions: `VectorTest`, `MatrixTest`, `BodyTest`, and `PlanetarySystemTest`. Each function has a corresponding `Console.WriteLine` statement to report the result. The bottom panel shows the `TERMINAL` output, which displays the results of running `mono Tests.exe`: `Vector test complete`, `Matrix test complete`, `Body test complete`, and `Planetary system test complete`. The status bar at the bottom indicates the file is on line 16, column 2, and the current branch is `git:(master)`.

```
1 using System;
2 using Structures;
3 using static Structures.Tests;
4
5 class Tests {
6     static void Main() {
7         if (VectorTest()) Console.WriteLine("Vector test complete");
8         else Console.WriteLine("Vector test failed");
9         if (MatrixTest()) Console.WriteLine("Matrix test complete");
10        else Console.WriteLine("Matrix test failed");
11        if (BodyTest()) Console.WriteLine("Body test complete");
12        else Console.WriteLine("Body test failed");
13        if (PlanetarySystemTest()) Console.WriteLine("Planetary system test complete");
14        else Console.WriteLine("Planetary system test failed");
15    }
16 }
```

```
> 0 [tinyclifford@ubuntu] ~/Projects/astrodynamics-sim $ mono Tests.exe
Vector test complete
Matrix test complete
Body test complete
Planetary system test complete
> 0 [tinyclifford@ubuntu] ~/Projects/astrodynamics-sim $
```

System Test documentation was taken as screen recordings and can be found at this link:

<https://www.dropbox.com/sh/3gn5k2rxhttp207k/AADQsOLul0vZfNrF8RVAU8pIa?dl=0>

## Evaluation

### Interview with Prospective User

An interview with Sophie Williams, A level physics student and prospective engineer.

“How was the user interface of the program?

Having the help button was extremely useful. The user interface was clear and concise, and simple enough to not overcrowd the screen. It’s meant to be a scientific demonstration so it was beneficial having all the different options on display.

What about in the simulation?

It would have been nice to see the values on display while running the simulation, even just saying how much it changes when the buttons are pressed, particularly the speed and trail length.

Did you enjoy the program?



## Astrodynamics Simulation

Yes! It was very educational, especially about binary star systems which I had never really considered before – it was fascinating to see the movement of different bodies in that binary star system. It was really useful having the preloaded systems to see what works and what doesn't before I tried to create my own system.

Do you understand orbital mechanics more now?

Definitely.

Would you recommend this program to other like minded people?

Yes! It's useful and helpful despite some inaccuracies at high simulation speed. Also fun to watch!

Do you think this would be a good tool for any of your teachers to teach about orbital mechanics?

I think students benefit greatly from seeing what the maths they are learning leads to, and believe this would be a great tool in being able to do that.

How convincing is the 3D projection?

Very. It's useful to be able to move the camera around, which gives a very good impression of the 3D projection onto the 2D screen. Of course it's impossible to get a realistic 3D projection until you have a hologram but it's doing pretty good for a computer program.

Any other comments?

I thought being able to change the color of planets gave it a nice touch."

### **Comparison against initial objectives:**

1. Accurately model simple Keplerian orbits – circular, elliptical, parabolic, and hyperbolic
  - Success. See eccentricity demo in Appendix 8
2. Provide a tool to add, remove, and edit bodies with classical orbital elements, and provide explanation.
  - Success. See System Test 1 and 2
3. Simulate motion in non-classical systems – e.g a rogue planet entering the solar system
  - Success. See RoguePlanet1 (System Test 9)
4. Render the 3D system into 2D convincingly.
  - Success. Agreed by interviewee.
5. Be able to place one or more cameras anywhere in the system and provide controls for zoom/rotation/etc.
  - Success. See System Tests 5, 6 and 7
6. Be able to pause the simulation and modify variables at any point.
  - Success. See System Test 4
7. Provide a capability to save and load systems to/from the hard drive.
  - Success. See System Test 3

### Conclusion

Overall, I think this project was a success. The objectives were met, the potential users were generally satisfied, and the program itself is fun to play with and very educational. As in any program, there are certainly things to be improved. Like a user suggested, the interface during the simulation could be more extensive, perhaps with a sidebar showing all the options with sliders etc. This might have been looked at more if I started the project again.

I am very pleased with the 3D projection, but it could be improved – maybe with realistic simulation of light sources via raytracing, or texture maps on the surface of planets. The mechanics, too – perturbations from radiation pressure and relativistic calculations could be added, or the numerical integration could be improved via a more efficient algorithm.

In total, however, I am very happy with the end result. I will certainly use it myself in the future, I hope others will too.

```

1  using System;
2  using System.Collections.Generic;
3  using static Program.Constants;
4  using System.Threading;
5  using System.Threading.Tasks;
6  using System.Linq;
7  namespace Structures
8  {
9      [Serializable()]
10     public class Vector3 {
11         // Simple 3-vector class, used for positions, velocities,
12         color, etc. // setters are required for deserialization but should not be
13         used outside class
14         public double x {get; set;}
15         public double y {get; set;}
16         public double z {get; set;}
17         public Vector3() {} // parameterless constructor for
18         serialization
19         public Vector3(double x, double y, double z) {
20             this.x = x;
21             this.y = y;
22             this.z = z;
23         }
24         // Immutable standard vectors
25         public static readonly Vector3 zero = new Vector3(0,0,0);
26         public static readonly Vector3 i = new Vector3(1,0,0);
27         public static readonly Vector3 j = new Vector3(0,1,0);
28         public static readonly Vector3 k = new Vector3(0,0,1);
29         public override String ToString() {
30             return $"Vector3({x},{y},{z})";
31         }
32         public static bool operator==(Vector3 a, Vector3 b) {
33             // Use inherited object null equality
34             if ((object)a == null || ((object)b == null)) return
35             (object)a == null && (object)b == null;
36             // otherwise return true if all components are within
37             10^-10
38             bool[] eq = new bool[3];
39             for (int i = 0; i < 3; i++) {
40                 double a1,b1;
41                 if (i == 0) {a1 = a.x; b1 = b.x;}
42                 else if (i == 1) {a1 = a.y; b1 = b.y;}
43                 else {a1 = a.z; b1 = b.z;}
44                 if (Math.Abs(a1) < 1e-2 || Math.Abs(b1) <
45                 1e-2) {
46                     eq[i] = Math.Abs(a1 - b1) < 1e-10;
47                 } else {
48                     eq[i] = Math.Abs((a1-b1)/a1) < 1e-10
49                     && Math.Abs((a1-b1)/b1) < 1e-10;
50                 }
51             }
52             return eq[0] && eq[1] && eq[2];
53         }
54         public static bool operator!=(Vector3 a, Vector3 b) {
55             // inverse of equality operator
56             return !(a == b);
57         }
58         public static Vector3 operator- (Vector3 a, Vector3 b) {
59             return new Vector3 (a.x-b.x,a.y-b.y,a.z-b.z);
60         }
61         public static Vector3 operator- (Vector3 a) {
62             return new Vector3(-a.x,-a.y,-a.z);
63         }
64         public static Vector3 operator+ (Vector3 a, Vector3 b) {
65             return new Vector3 (a.x+b.x,a.y+b.y,a.z+b.z);
66         }
67     }
68 }

```

```

61         }
62         public static Vector3 operator* (double a, Vector3 b) {
63             return new Vector3 (a*b.x,a*b.y,a*b.z);
64         }
65         public static Vector3 operator/ (Vector3 a, double b) {
66             return new Vector3 (a.x/b,a.y/b,a.z/b);
67         }
68         public static double dot(Vector3 a, Vector3 b) {
69             // This could be overloaded to operator*, but an
explicit function increases readability.
70             return a.x*b.x + a.y*b.y + a.z*b.z;
71         }
72         public static Vector3 cross(Vector3 a, Vector3 b) {
73             return new Vector3(
74                 a.y*b.z - a.z*b.y,
75                 a.z*b.x - a.x*b.z,
76                 a.x*b.y - a.y*b.x
77             );
78         }
79         public static double Magnitude(Vector3 v) {
80             // Pythagorean Theorem
81             return Math.Sqrt(Math.Pow(v.x,2)+Math.Pow(v.y,2)
+Math.Pow(v.z,2));
82         }
83         public static Vector3 Unit(Vector3 v) {
84             // Throw exception if v is an invalid value
85             if (v == Vector3.zero) {
86                 throw new DivideByZeroException("Cannot take
unit of zero vector");
87             }
88             return v / Vector3.Magnitude(v);
89         }
90         public static double UnitDot(Vector3 a, Vector3 b) {
91             // The dot of the unit vectors
92             return Vector3.dot(Vector3.Unit(a),Vector3.Unit(b));
93         }
94         public static Vector3 Log(Vector3 v, double b = Math.E) {
95             // Polar logarithm (radius is logged, direction is
consistent)
96             var polar = CartesianToPolar(v);
97             var log_polar = new Vector3 (Math.Log
(polar.x,b),polar.y,polar.z);
98             var log = PolarToCartesian(log_polar);
99             return log;
100         }
101         public static Vector3 LogByComponent(Vector3 v, double b =
Math.E) {
102             // Cartesian Logarithm, all components are logged
103             var r = new Vector3(0,0,0);
104             if (v.x < 0) r.x = -Math.Log(-v.x,b);
105             else if (v.x != 0) r.x = Math.Log(v.x,b);
106             if (v.y < 0) r.y = -Math.Log(-v.y,b);
107             else if (v.y != 0) r.y = Math.Log(v.y,b);
108             if (v.z < 0) r.z = -Math.Log(-v.z,b);
109             else if (v.z != 0) r.z = Math.Log(v.z,b);
110             return r;
111         }
112         public static Vector3 CartesianToPolar(Vector3 v) {
113             // ISO Convention
114             var r = Vector3.Magnitude(v);
115             var theta = Math.Acos(Vector3.UnitDot(v,Vector3.k));
116             var phi = Math.Acos(Vector3.UnitDot(new Vector3
(v.x,v.y,0),Vector3.i));
117             if (v.y < 0) phi = -phi;
118             return new Vector3(r,theta,phi);
119         }

```

```

120         public static Vector3 PolarToCartesian(Vector3 v) {
121             // ISO Convention
122             return Matrix3.ZRotation(v.z) * Matrix3.YRotation
(v.y) * (v.x*Vector3.k);
123         }
124     }
125 }
126 public class Matrix3 {
127     // the fields describe the rows. Using Vector3s makes Matrix-
Vector Multiplication
128     // (which is the most useful operation) simpler, since then
Vector3.dot can be used
129     public Vector3 x {get;}
130     public Vector3 y {get;}
131     public Vector3 z {get;}
132     public Matrix3(Vector3 x, Vector3 y, Vector3 z) {
133         this.x = x;
134         this.y = y;
135         this.z = z;
136     }
137     public override String ToString() {
138         return $"Matrix3( {x.x} {x.y} {x.z}\n          {y.x}
{y.y} {y.z}\n          {z.x} {z.y} {z.z} )";
139     }
140     public static Matrix3 XRotation(double x) {
141         return new Matrix3 (
142             new Vector3(1,0,0),
143             new Vector3(0,Math.Cos(x),Math.Sin(x)),
144             new Vector3(0,-Math.Sin(x),Math.Cos(x))
145         );
146     }
147     public static Matrix3 YRotation(double y) {
148         return new Matrix3 (
149             new Vector3(Math.Cos(y),0,Math.Sin(y)),
150             new Vector3(0,1,0),
151             new Vector3(-Math.Sin(y),0,Math.Cos(y))
152         );
153     }
154     public static Matrix3 ZRotation(double z) {
155         return new Matrix3 (
156             new Vector3(Math.Cos(z),-Math.Sin(z),0),
157             new Vector3(Math.Sin(z),Math.Cos(z),0),
158             new Vector3(0,0,1)
159         );
160     }
161     public static Matrix3 ExtrinsicZYXRotation(double x, double
y, double z) {
162         return XRotation(x)*YRotation(y)*ZRotation(z);
163     }
164     public static Matrix3 ExtrinsicZYXRotation(Vector3 v) {
165         return XRotation(v.x)*YRotation(v.y)*ZRotation(v.z);
166     }
167     public static Matrix3 IntrinsicZYXRotation(double x, double
y, double z) {
168         return ZRotation(z)*YRotation(y)*XRotation(x);
169     }
170     public static Matrix3 IntrinsicZYXRotation(Vector3 v) {
171         return ZRotation(v.z)*YRotation(v.y)*XRotation(v.x);
172     }
173     public static bool operator== (Matrix3 a, Matrix3 b) {
174         // Use vector equality
175         return a.x == b.x && a.y == b.y && a.z == b.z;
176     }
177     public static bool operator!= (Matrix3 a, Matrix3 b) {
178         return !(a == b);
179     }

```

```

180
181         public static Matrix3 operator+ (Matrix3 a, Matrix3 b) {
182             // Add component-wise
183             return new Matrix3(
184                 a.x + b.x,
185                 a.y + b.y,
186                 a.z + b.z
187             );
188         }
189         public static Vector3 operator* (Matrix3 m, Vector3 v) {
190             // Using the fact that a matrix (lxn) multiplied by a
191             // (nx1) is equivalent to the dot of two n-vectors
192             return new Vector3(
193                 Vector3.dot(m.x,v),
194                 Vector3.dot(m.y,v),
195                 Vector3.dot(m.z,v)
196             );
197         }
198         public static Matrix3 operator* (double d, Matrix3 m) {
199             // multiply each component by d
200             return new Matrix3(
201                 d * m.x,
202                 d * m.y,
203                 d * m.z
204             );
205         }
206         public static Matrix3 operator/ (Matrix3 m, double d) {
207             // raise exception on invalid value
208             if (d == 0) throw new DivideByZeroException("Matrix
209             Division By Zero");
210             else return (1/d) * m;
211         }
212         public static Matrix3 operator* (Matrix3 l, Matrix3 r) {
213             // Finding a new matrix of the transpose of r
214             // converts it from row vectors to column vectors
215             // so we can use the dot product to find each value
216             var r_t = Matrix3.Transpose(r);
217             return new Matrix3 (
218                 new Vector3(
219                     Vector3.dot(l.x,r_t.x),
220                     Vector3.dot(l.x,r_t.y),
221                     Vector3.dot(l.x,r_t.z)
222                 ),
223                 new Vector3(
224                     Vector3.dot(l.y,r_t.x),
225                     Vector3.dot(l.y,r_t.y),
226                     Vector3.dot(l.y,r_t.z)
227                 ),
228                 new Vector3(
229                     Vector3.dot(l.z,r_t.x),
230                     Vector3.dot(l.z,r_t.y),
231                     Vector3.dot(l.z,r_t.z)
232                 )
233             );
234         }
235         public static double Determinant(Matrix3 m) {
236             return m.x.x * (m.y.y*m.z.z - m.y.z*m.z.y)
237                 -m.x.y * (m.y.x*m.z.z - m.y.z*m.z.x)
238                 +m.x.z * (m.y.x*m.z.y - m.y.y*m.z.x);
239         }
240         public static Matrix3 Transpose(Matrix3 m) {
241             return new Matrix3(
242                 new Vector3(m.x.x,m.y.x,m.z.x),
243                 new Vector3(m.x.y,m.y.y,m.z.y),
244                 new Vector3(m.x.z,m.y.z,m.z.z)
245             );

```

```
243     }
244     public static Matrix3 Adjugate(Matrix3 m) {
245         return new Matrix3(
246             new Vector3(m.x.x, -m.y.x, m.z.x),
247             new Vector3(-m.x.y, m.y.y, -m.z.y),
248             new Vector3(m.x.z, -m.y.z, m.z.z)
249         );
250     }
251     public static Matrix3 Minor(Matrix3 m) {
252         return new Matrix3(
253             new Vector3(
254                 (m.y.y*m.z.z - m.y.z*m.z.y),
255                 (m.y.x*m.z.z - m.y.z*m.z.x),
256                 (m.y.x*m.z.y - m.y.y*m.z.x)
257             ),
258             new Vector3(
259                 (m.x.y*m.z.z - m.x.z*m.z.y),
260                 (m.x.x*m.z.z - m.x.z*m.z.x),
261                 (m.x.x*m.z.y - m.x.y*m.z.x)
262             ),
263             new Vector3(
264                 (m.x.y*m.y.z - m.x.z*m.y.y),
265                 (m.x.x*m.y.z - m.x.z*m.y.x),
266                 (m.x.x*m.y.y - m.x.y*m.y.x)
267             )
268         );
269     }
270     public static Matrix3 Inverse(Matrix3 m) {
271         if (Matrix3.Determinant(m) == 0) throw new
DivideByZeroException("Singular Matrix");
272         Matrix3 A = Matrix3.Adjugate(Matrix3.Minor(m));
273         return (1/Matrix3.Determinant(m)) * A;
274     }
275 }
276 }
```

```

1  using System;
2  using System.Collections.Generic;
3  using static Program.Constants;
4  using System.Threading;
5  using System.Threading.Tasks;
6  using System.Linq;
7  namespace Structures
8  {
9      [Serializable()]
10     public class Body : ICloneable {
11         public string name {get; set;}
12         public Body parent {get; set;}
13         public double stdGrav {get; set;} // standard gravitational
parameter
14         public double radius {get; set;}
15         public Vector3 position {get; set;} = Vector3.zero;
16         public Vector3 velocity {get; set;} = Vector3.zero;
17         public Vector3 color {get; set;} = new Vector3(1,1,1);
18         public Body() {} // parameterless constructor for
serialisation
19         public Body (Body parent, OrbitalElements elements) {
20             // First check the values are reasonable. If parent
== null it is assumed that
21             // position and velocity are set explicitly, and this
constructor is not used
22             if (parent == null) return;
23             this.parent = parent;
24             if (elements.eccentricity < 0
25                 || elements.semilatusrectum < 0
26                 || elements.inclination < 0
27                 || elements.inclination > Math.PI
28                 || elements.ascendingNodeLongitude < 0
29                 || elements.ascendingNodeLongitude >= 2*Math.PI
30                 || elements.periapsisArgument < 0
31                 || elements.periapsisArgument >= 2*Math.PI
32                 || elements.trueAnomaly < 0
33                 || elements.trueAnomaly >= 2*Math.PI
34             ){
35                 // Throw an exception if the arguments are
out of bounds
36                 throw new ArgumentException();
37             }
38             // working in perifocal coordinates (periapsis along
the x axis, orbit in the x,y plane):
39             double mag_peri_radius = elements.semilatusrectum/(1
+elements.eccentricity*Math.Cos(elements.trueAnomaly));
40             Vector3 peri_radius = mag_peri_radius*new Vector3
(Math.Cos(elements.trueAnomaly),Math.Sin(elements.trueAnomaly),0);
41             Vector3 peri_velocity = Math.Sqrt(parent.stdGrav/
elements.semilatusrectum)
42                                     * new
Vector3(
43             Math.Sin(elements.trueAnomaly),
44             Math.Cos(elements.trueAnomaly) + elements.eccentricity,
45             0
46                                     );
47             // useful constants to setup transformation matrix
48             var sini = Math.Sin(elements.inclination); // i <-
inclination
49             var cosi = Math.Cos(elements.inclination);
50             var sino = Math.Sin
(elements.ascendingNodeLongitude); // capital omega <- longitude of ascending
node

```



```

51         var coso = Math.Cos(elements.ascendingNodeLongitude);
52         var sinw = Math.Sin(elements.periapsisArgument); //
    omega <- argument of periapsis
53         var cosw = Math.Cos(elements.periapsisArgument);
54         // Transform perifocal coordinates to i,j,k
    coordinates
55         Matrix3 transform = new Matrix3(
56             new Vector3(
57                 coso*cosw - sino*sinw*cosi,
58                 -coso*sinw-sino*cosw*cosi,
59                 sino*sini
60             ),
61             new Vector3(
62                 sino*cosw+coso*sinw*cosi,
63                 -sino*sinw+coso*cosw*cosi,
64                 -coso*sini
65             ),
66             new Vector3(
67                 sinw*sini,
68                 cosw*sini,
69                 cosi
70             )
71         );
72         // add the parent's position and velocity since that
    could be orbiting something too
73         this.position = transform*peri_radius +
    parent.position;
74         this.velocity = transform*peri_velocity +
    parent.velocity;
75     }
76     public double HillRadius() {
77         // This is the maximum distance anything can
    reasonably orbit at.
78         // It would normally depend on the bodies nearby, but
    we'll just do something simple
79         // which is roughly accurate for bodies in the solar
    system.
80         return this.stdGrav * 1e-6;
81     }
82     public object Clone() {
83         return new Body {
84             name = this.name,
85             parent = this.parent,
86             stdGrav = this.stdGrav,
87             radius = this.radius,
88             position = this.position,
89             velocity = this.velocity,
90             color = this.color
91         };
92     }
93 }

```

```

1  using System;
2  using System.Collections.Generic;
3  using static Program.Constants;
4  using System.Threading;
5  using System.Threading.Tasks;
6  using System.Linq;
7  namespace Structures
8  {
9      internal class FundamentalVectors {
10         // The fundamental vectors of an orbit. Used by
11         OrbitalElements
12         public Vector3 angularMomentum {get; set;}
13         public Vector3 eccentricity {get; set;}
14         public Vector3 node {get; set;}
15         public FundamentalVectors(Vector3 position, Vector3 velocity,
16         double stdGrav) {
17             this.angularMomentum = Vector3.cross
18             (position,velocity);
19             this.node = Vector3.cross
20             (Vector3.k,this.angularMomentum);
21             var mag_r = Vector3.Magnitude(position);
22             var mag_v = Vector3.Magnitude(velocity);
23             this.eccentricity = (1/stdGrav)*((Math.Pow(mag_v,2) -
24             stdGrav/mag_r)*position - Vector3.dot(position,velocity)*velocity);
25         }
26         public override String ToString() {
27             return $"Angular Momentum: {angularMomentum.ToString
28             ()}\nEccentricity: {eccentricity.ToString()}\nNode: {node.ToString()}";
29         }
30     }
31     public class OrbitalElements {
32         // The six classical orbital elements
33         public double semilatusrectum {get; set;}
34         public double eccentricity {get; set;}
35         protected double _inclination;
36         public double inclination {
37             get {
38                 return _inclination;
39             } set {
40                 _inclination = value*Math.PI;
41             }
42         }
43         protected double _ascendingNodeLongitude;
44         public double ascendingNodeLongitude {
45             get {
46                 return _ascendingNodeLongitude;
47             } set {
48                 _ascendingNodeLongitude = value%(2*Math.PI);
49             }
50         }
51         protected double _periapsisArgument;
52         public double periapsisArgument {
53             get {
54                 return _periapsisArgument;
55             } set {
56                 _periapsisArgument = value%(2*Math.PI);
57             }
58         }
59         protected double _trueAnomaly;
60         public double trueAnomaly {
61             get {
62                 return _trueAnomaly;
63             } set {
64                 _trueAnomaly = value%(2*Math.PI);
65             }
66         }
67     }
68 }

```

```

61         }
62         public OrbitalElements() {} // Parameterless constructor for
serialisation
63         public OrbitalElements(Vector3 position, Vector3 velocity,
double stdGrav) {
64             // stdGrav is the gravitational parameter of the
parent body
65             var fVectors = new FundamentalVectors
(position,velocity,stdGrav);
66             this.eccentricity = Vector3.Magnitude
(fVectors.eccentricity);
67             this.semilatusrectum = Math.Pow(Vector3.Magnitude
(fVectors.angularMomentum),2)/stdGrav;
68             this.inclination = Math.Acos
(fVectors.angularMomentum.z/Vector3.Magnitude(fVectors.angularMomentum)); //
0 <= i <= 180deg
69             double cosAscNodeLong = fVectors.node.x/
Vector3.Magnitude(fVectors.node);
70             if (fVectors.node.y >= 0) this.ascendingNodeLongitude
= Math.Acos(cosAscNodeLong);
71             else this.ascendingNodeLongitude = 2*Math.PI -
Math.Acos(cosAscNodeLong);
72             double cosAnomaly = 0;
73             try {
74                 double cosPeriArg = Vector3.UnitDot
(fVectors.node,fVectors.eccentricity);
75                 if (fVectors.eccentricity.z >= 0)
this.periapsisArgument = Math.Acos(cosPeriArg);
76                 else this.periapsisArgument = 2*Math.PI -
Math.Acos(cosPeriArg);
77                 cosAnomaly = Vector3.UnitDot
(fVectors.eccentricity,position);
78             } catch (DivideByZeroException) {
79                 // This will be dealt with along with
extremely small values below
80             }
81             if (this.eccentricity < 1e-10 ) {
82                 // acceptable error, the orbit has no
periapsis
83                 this.eccentricity = 0;
84                 this.periapsisArgument = 0;
85                 // we assume the periapsis is at the node
vector
86                 if (Vector3.Magnitude(fVectors.node) < 1e-10)
{
87                     // but if the node vector also does
not exist we assume the i vector
88                     cosAnomaly = Vector3.UnitDot
(Vector3.i,position);
89                 } else {
90                     cosAnomaly = Vector3.UnitDot
(fVectors.node, position);
91                 }
92             }
93             if (Vector3.UnitDot(position,velocity) >= 0)
this.trueAnomaly = Math.Acos(cosAnomaly);
94             else this.trueAnomaly = 2*Math.PI - Math.Acos
(cosAnomaly);
95             if (Math.Abs(fVectors.angularMomentum.x/
fVectors.angularMomentum.z) < 1e-10
96                 && Math.Abs(fVectors.angularMomentum.y/
fVectors.angularMomentum.z) < 1e-10) {
97                 // acceptable error, the orbit is not inclined
98                 this.ascendingNodeLongitude = 0;
99             }
100         }

```

---

```
101         }  
102     }
```

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using static Program.Constants;
5  using System.Threading;
6  using System.Threading.Tasks;
7  using System.Linq;
8  namespace Structures
9  {
10     public class PlanetarySystem : IEnumerable<Body> {
11         protected bool running = false;
12         protected List<Body> bodies;
13         public List<int> centers {get; set;} = new List<int>();
14         protected int center_index = -1; // -1 indicates space is not
locked
15         public PlanetarySystem(List<Body> bodies = null) {
16             if (bodies == null) this.bodies = new List<Body>();
17             else this.bodies = bodies;
18         }
19         public Body this[int key] {
20             get {
21                 return this.bodies[key];
22             }
23         }
24         public IEnumerator<Body> GetEnumerator() { return
this.bodies.GetEnumerator(); }
25         IEnumerator IEnumerable.GetEnumerator() { return
this.bodies.GetEnumerator(); }
26         public int Count {
27             get {
28                 return this.bodies.Count;
29             }
30         }
31         public void Add(Body body) {
32             bodies.Add(body);
33         }
34         public Vector3 Barycenter() {
35             Vector3 weighted_center = Vector3.zero;
36             double mu_total = 0;
37             foreach (Body b in this) {
38                 mu_total += b.stdGrav;
39                 weighted_center += b.stdGrav*b.position;
40             }
41             return weighted_center/mu_total;
42         }
43         public void IterateCenter() {
44             this.center_index += 1;
45             if (this.center_index >= this.centers.Count) {
46                 this.center_index = -1;
47             }
48         }
49         public Vector3 origin {
50             get {
51                 if (this.center_index == -1) return
this.Barycenter();
52                 else return this[this.centers
[this.center_index]].position;
53             }
54         }
55         protected Vector3[] GetAcceleration() {
56             Vector3[] acceleration = new Vector3[this.Count];
57             // Initialise our array to Vector3.zero, since the
default is a null pointer.
58             Parallel.For (0, this.Count, i => {
59                 acceleration[i] = Vector3.zero;
60             });

```

```

61         for (int i = 0; i < this.Count; i++) {
62             // We will need the index later so foreach is
not possible
63             Body body1 = this[i];
64             for (int j = i + 1; j < this.Count; j++) {
65                 Body body2 = this[j]; // Again here
66                 // The magnitude of the force,
multiplied by G, = %mu_1 * %mu_2 / r^2
67                 double mag_force_g = body1.stdGrav *
body2.stdGrav / Math.Pow(Vector3.Magnitude(body1.position - body2.position),2);
68                 // We lost direction in the previous
calculation (since we had to square the
69                 vector), but we need it.
Vector3 direction = Vector3.Unit
(body1.position - body2.position);
70                 // since acceleration is F/m, and we
have G*F and G*m, we can find an acceleration vector easily
71                 Vector3 acceleration1 = mag_force_g *
-direction / body1.stdGrav;
72                 Vector3 acceleration2 = mag_force_g *
direction / body2.stdGrav;
73                 acceleration[i] += acceleration1;
74                 acceleration[j] += acceleration2;
75             }
76         }
77         return acceleration;
78     }
79     protected void TimeStep(double step) {
80         var acceleration = this.GetAcceleration();
81         for (int i = 0; i < acceleration.Length; i++) {
82             Body body = this[i];
83             Vector3 a = acceleration[i];
84             body.position += step*body.velocity + Math.Pow
(step,2)*a/2;
85             body.velocity += step*a;
86         }
87     }
88     public void StartAsync(double step = 1) {
89         Task.Run(() => Start(step));
90     }
91     public void Start(double step = 1) {
92         this.running = true;
93         while (running) this.TimeStep(step);
94     }
95     public void Stop() {
96         this.running = false;
97     }
98 }
99 }

```

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using Gtk;
5  using Cairo;
6  using Structures;
7  using System.Threading;
8  using System.Threading.Tasks;
9  using static Program.Constants;
10 namespace Graphics {
11     class SystemView : DrawingArea {
12         public Camera camera {get; set;} //= new Camera
(1,Vector3.zero);
13         public double radius_multiplier {get; set;} = 1;
14         public int line_max {get; set;} = 100;
15         public double bounds_multiplier {get; set;} = 1;//0.25;
16         protected PlanetarySystem sys;
17         protected readonly double LINE_MULTIPLIER = 0.8;
18         protected bool playing = false;
19         protected List<Vector3>[] paths;
20         protected int[] order;
21         protected double max = 0;
22         public SystemView(PlanetarySystem sys) {
23             this.sys = sys;
24             this.camera = new Camera(sys.Max(b =>
Vector3.Magnitude(b.position - sys.origin)),Vector3.zero);
25             SetMax();
26         }
27         public void SetMax() {
28             order = new int[sys.Count];
29             for (int i = 0; i < sys.Count; i++) order[i] = i;
30             max = 0;
31             foreach (Body b in sys) {
32                 var v = camera.TransformProjection
(camera.Transform(b.position - sys.origin));
33                 var p = Vector3.Magnitude(new Vector3
(v.x,v.y,0));
34                 if (p > max) {
35                     max = p;
36                 }
37             }
38         }
39         public void ClearPaths() {
40             this.paths = new List<Vector3>[sys.Count];
41             for (int i = 0; i < sys.Count; i++) {
42                 this.paths[i] = new List<Vector3>();
43             }
44         }
45         public void Play(int interval) {
46             playing = true;
47             while (playing) {
48                 this.QueueDraw();
49                 Thread.Sleep(interval);
50             }
51         }
52         public void PlayAsync(int interval) {
53             Task.Run(() => Play(interval));
54         }
55         public void Stop() {
56             playing = false;
57         }
58         protected override bool OnDrawn (Cairo.Context ctx) {
59             // color the screen black
60             ctx.SetSourceRGB(0,0,0);
61             ctx.Paint();
62         }

```

```

63         // Normally (0,0) is in the corner, but we want it in
the middle, so we must translate:
64         ctx.Translate(AllocatedWidth/2,AllocatedHeight/2);
65         var bounds = bounds_multiplier * max * new Vector3
(1,1,1);
66         // we care about the limiting factor, since most
orbits will be bounded roughly by a square
67         // but screens are rectangular
68         var scale = Math.Min((AllocatedWidth/2)*bounds.x,
(AllocatedHeight/2)/bounds.y);
69         ctx.Scale(scale,scale);
70
71         if (paths == null) {
72             this.ClearPaths();
73         }
74         order = order.OrderByDescending(x => Vector3.Magnitude
(sys[x].position - camera.position)).ToArray();
75         for (int i = 0; i < sys.Count; i++) {
76             var body = sys[order[i]];
77             var cl = body.color;
78             ctx.SetSourceRGB (cl.x,cl.y,cl.z);
79
80             var T = camera.Transform(body.position -
sys.origin); //camera.position); // - camera.Transform(sys.origin);
81
82             var r = radius_multiplier *
camera.TransformProjectionRadius(T,body.radius); //body.radius;
83             var pos = camera.TransformProjection(T);
84             ctx.Arc(pos.x,pos.y,r,0,2*Math.PI);
85             ctx.Fill();
86             Vector3 lastPath;
87             try {
88                 lastPath = camera.TransformProjection
(camera.Transform(paths[order[i]][0]));
89             } catch (ArgumentOutOfRangeException) {
90                 lastPath = Vector3.zero;
91             }
92             ctx.LineWidth = Math.Min(LINE_MULTIPLIER *
radius_multiplier * body.radius, LINE_MULTIPLIER*r);
93             foreach (Vector3 p in paths[order[i]]) {
94                 pos = camera.TransformProjection
(camera.Transform(p));
95                 ctx.MoveTo(lastPath.x,lastPath.y);
96                 ctx.LineTo(pos.x,pos.y);
97                 ctx.Stroke();
98                 lastPath = pos;
99             }
100             paths[order[i]].Add(body.position -
sys.origin);
101             if (paths[order[i]].Count > line_max) paths
[order[i]] = paths[order[i]].TakeLast(line_max).ToList();
102         }
103         return true;
104     }
105 }
106 }

```



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using Gtk;
5  using Cairo;
6  using Structures;
7  using System.Threading;
8  using System.Threading.Tasks;
9  using static Program.Constants;
10 namespace Graphics {
11     class Camera {
12         public Vector3 position {get; protected set;}
13         public Vector3 angle {get; protected set;}
14         protected double focalLength; // {get; protected set;} = 50*AU;
15         public Camera(double distance, Vector3 angle) {
16             // the camera always "points" to the origin
17             this.angle = angle;
18             position = Matrix3.IntrinsicZYXRotation(angle)*new
Vector3(0,0,-distance);
19             focalLength = distance;
20         }
21         public Vector3 Transform(Vector3 position) {
22             return Matrix3.ExtrinsicZYXRotation(this.angle)*
(position); // - this.position;
23         }
24         public Vector3 TransformProjection(Vector3 T) {
25             var z = T.z + focalLength;
26             return (focalLength/z)*T;
27         }
28         public double TransformProjectionRadius(Vector3 T, double r) {
29             return r*Math.Atan(r/(T.z+focalLength))/Math.Atan(r/
focalLength);
30         }
31     }
32 }
33 }
```

```

1  using System;
2  using System.IO;
3  using System.Linq;
4  using System.Collections.Generic;
5  using Gtk;
6  using Cairo;
7  using static Program.Program;
8  using static Program.Constants;
9  using Structures;
10 namespace UI {
11     public class Menu : Window {
12         protected VBox containerbox;
13         protected VBox controlbox;
14         protected ScrolledWindow systemscrollbox;
15         protected VBox systembox;
16         protected HBox donebox;
17         protected Scale TimestepScale;
18         protected Scale RScale;
19         protected Scale LineScale;
20         protected ComboBoxText BodyCombo;
21         public Button loadButton {get; set;}
22         protected Entry filename;
23         protected readonly String SYSTEM_DIRECTORY = "ExampleSystems";
24         internal List<BodyBox> new_bodies {get; set;} = new List<BodyBox>();
25         public SaveData temp_savedata {get; set;} = null;
26         protected static List<bool> centers = new List<bool>();
27
28         public Menu(Gtk.WindowType s = Gtk.WindowType.Toplevel) : base(s) {
29             this.SetDefaultSize(300,400);
30             this.DeleteEvent += delegate { Application.Quit (); };
31             containerbox = new VBox(homogeneous: false, spacing: 3);
32             controlbox = new VBox(homogeneous: false, spacing: 3);
33             systemscrollbox = new ScrolledWindow();
34             systembox = new VBox(homogeneous: false, spacing: 3);
35             donebox = new HBox(homogeneous: false, spacing: 3);
36
37             var l1 = new Label("Mechanics Timestep");
38             var l2 = new Label("Planetary Radii Multiplier");
39             var l3 = new Label("Orbit Trail Length");
40             TimestepScale = new Scale(Orientation.Horizontal, 0.1,1000,0.1);
41             TimestepScale.Value = 50;
42             RScale = new Scale(Orientation.Horizontal, 1, 1000, 1);
43             RScale.Value = 100;
44             LineScale = new Scale(Orientation.Horizontal, 50, 1000, 1);
45             LineScale.Value = 100;
46
47             var addBox = new HBox();
48             var addButton = new Button("Add");
49             addButton.Clicked += new EventHandler(OnAddClick);
50             var filenameText = new Label("Save File: ");
51             filename = new Entry();
52             var saveButton = new Button("Save");
53             saveButton.Clicked += new EventHandler(OnSaveClick);
54             loadButton = new Button("Load");
55             loadButton.Clicked += new EventHandler(OnLoadClick);
56             var helpButton = new Button("?");
57             helpButton.Clicked += new EventHandler(OnHelpClick);
58             BodyCombo = new ComboBoxText();
59             BodyCombo.AppendText("Custom");
60             foreach (Body b in Examples.solar_system_bodies) {
61                 BodyCombo.AppendText(b.name);
62             }
63             BodyCombo.Active = 0; // Default to Custom body
64             addBox.PackStart(BodyCombo, true, false, 3);
65             addBox.PackStart(addButton, true, false, 3);
66             addBox.PackStart(filenameText, true, false, 3);

```

```

67         addBox.PackStart(filename, true, false, 3);
68         addBox.PackStart(saveButton, true, false, 3);
69         addBox.PackStart(loadButton, true, false, 3);
70         addBox.PackStart(helpButton, true, false, 3);
71         var doneButton = new Button("Done");
72         doneButton.Clicked += new EventHandler (OnDoneClick);
73         var exitButton = new Button("Exit");
74         exitButton.Clicked += new EventHandler(delegate{
75             Application.Quit();
76         });
77
78         var optionsbox = new HBox(homogeneous: false, spacing: 3);
79         var optionbox1 = new VBox(homogeneous: false, spacing: 3);
80         var optionbox2 = new VBox(homogeneous: false, spacing: 3);
81         var optionbox3 = new VBox(homogeneous: false, spacing: 3);
82         optionbox1.PackStart(l1, true, true, 3);
83         optionbox1.PackStart(TimestepScale, true, true, 3);
84         optionbox2.PackStart(l2, true, true, 3);
85         optionbox2.PackStart(RScale, true, true, 3);
86         optionbox3.PackStart(l3, true, true, 3);
87         optionbox3.PackStart(LineScale, true, true, 3);
88         optionsbox.PackStart(optionbox1, true, true, 3);
89         optionsbox.PackStart(optionbox2, true, true, 3);
90         optionsbox.PackStart(optionbox3, true, true, 3);
91
92         controlbox.PackStart(optionsbox, false, false, 3);
93         controlbox.PackStart(addButton, false, false, 3);
94         controlbox.PackStart(addBox, false, false, 3);
95         systemscrollbox.Add(systembox);
96         donebox.PackStart(doneButton, true, true, 3);
97         donebox.PackStart(exitButton, true, true, 3);
98
99
100        containerbox.PackStart(controlbox, false, false, 3);
101        containerbox.PackStart(systemscrollbox, true, true, 3);
102        containerbox.PackStart(donebox, false, false, 3);
103        this.Add(containerbox);
104        this.ShowAll();
105    }
106    protected void OnDoneClick(object obj, EventArgs args) {
107        if (new_bodies.Count < 2) {
108            Message("An empty system is not very interesting!");
109            return;
110        }
111        try {
112            Program.Program.CustomBodies.Clear();
113            Program.Program.CustomCenters.Clear();
114            centers.Clear();
115            foreach (BodyBox b in new_bodies) {
116                b.Set();
117                Program.Program.CustomBodies.Add(b.body);
118                centers.Add(b.CenterButton.Active);
119            }
120            Program.Program.CustomCenters = centers;
121            Program.Program.radius_multiplier = RScale.Value;
122            Program.Program.line_max = (int)LineScale.Value;
123            Program.Program.timestep = TimestepScale.Value;
124            Program.Program.StartSimulation();
125            this.Destroy();
126        } catch (Exception e) {
127            Message("I'm sorry, something went wrong but I don't know
128            what. \nIf you can find a bored developer, show him this stack trace:\n" +
129            e.Message + e.StackTrace);
130        }
131    }

```

```

131         protected void OnAddClick(object obj, EventArgs args) {
132
133             var bodyBox = new BodyBox(menu: this, homogeneous: false,
spacing: 3);
134             String bString = BodyCombo.ActiveText;
135             if (bString != "Custom") {
136                 var body = Examples.solar_system.First(b => b.name ==
bString);
137                 if (!(body.parent == null || new_bodies.Exists(b =>
b.name.Text == body.parent.name))) {
138                     body = Examples.solar_system_bodies.First(b => b.name ==
bString);
139                 }
140                 bodyBox.body = body;
141                 bodyBox.ReverseSet();
142             }
143
144             bodyBox.name.Text = BodyCombo.ActiveText;
145             systembox.PackStart(bodyBox, true, true, 3);
146             new_bodies.Add(bodyBox);
147             foreach (BodyBox b in new_bodies) {
148                 b.ResetParents();
149             }
150             this.ShowAll();
151         }
152         protected void OnSaveClick(object obj, EventArgs args) {
153             if (filename.Text == "") {
154                 Message("Please enter a filename");
155                 return;
156             }
157             System.Xml.Serialization.XmlSerializer writer =
158                 new System.Xml.Serialization.XmlSerializer(typeof(SaveData));
159             if (File.Exists(Environment.CurrentDirectory + "/" +
filename.Text + ".xml")) {
160                 File.Delete(Environment.CurrentDirectory + "/" +
filename.Text + ".xml");
161             }
162             FileStream file = File.Create(
163                 Environment.CurrentDirectory + "/" + filename.Text + ".xml");
164             var bodies = new List<Body>();
165             if (centers == null) centers = new List<bool>();
166             centers.Clear();
167             var elements = new List<OrbitalElements>();
168             foreach (BodyBox b in new_bodies) {
169                 b.Set();
170                 bodies.Add(b.body);
171                 centers.Add(b.CenterButton.Active);
172                 elements.Add(new OrbitalElements() {
173                     semilatusrectum = b.SLRScale.Value*AU,
174                     eccentricity = b.EScale.Value,
175                     inclination = b.IncScale.Value*deg,
176                     ascendingNodeLongitude = b.ANLScale.Value*deg,
177                     periapsisArgument = b.PAScale.Value*deg,
178                     trueAnomaly = b.TAScale.Value*deg
179                 });
180             }
181             var data = new SaveData() {
182                 bodies = bodies,
183                 elements = elements,
184                 timestep = TimestepScale.Value,
185                 centers = centers,
186                 radius_multiplier = RScale.Value,
187                 line_max = LineScale.Value,
188             };
189             writer.Serialize(file, data);
190             file.Close();

```

```

191     }
192     protected void OnLoadClick(object obj, EventArgs args) {
193         System.Xml.Serialization.XmlSerializer reader =
194             new System.Xml.Serialization.XmlSerializer(typeof(SaveData));
195         SaveData data = new SaveData(); // To prevent compiler error
196         if (temp_savedata != null) {
197             data = temp_savedata;
198             temp_savedata = null;
199         } else {
200             try {
201                 var file = new StreamReader(Environment.CurrentDirectory
+ "/" + filename.Text + ".xml");
202                 data = (SaveData)reader.Deserialize(file);
203             } catch (IOException) {
204                 // Try in the system directory
205                 try {
206                     var file = new StreamReader
(Environment.CurrentDirectory + "/" + SYSTEM_DIRECTORY + "/" +
filename.Text + ".xml");
207                     data = (SaveData)reader.Deserialize(file);
208                 } catch (IOException) {
209                     Message("The specified file could not be found. Check
that the name is spelt correctly and that it is in the correct directory");
210                     // cannot deserialize, exit
211                     return;
212                 }
213             } catch (InvalidOperationException) {
214                 Message("The file is not a valid save file of this
project");
215                 // cannot deserialize, exit
216                 return;
217             }
218         }
219         RScale.Value = data.radius_multiplier;
220         LineScale.Value = data.line_max;
221         TimestepScale.Value = data.timestep;
222         new_bodies.Clear();
223         foreach (Widget w in systembox.Children) {
224             if (w is BodyBox) systembox.Remove (w);
225         }
226         for (int i = 0; i < data.bodies.Count; i++) {
227             var bbox = new BodyBox(menu: this, homogeneous: false,
spacing: 3) {
228                 body = data.bodies[i],
229             };
230             bbox.CenterButton.Active = data.centers[i];
231             if (data.elements != null && data.elements.Count != 0) {
232                 bbox.SetElements(data.elements[i]);
233                 bbox.ReverseSet(false);
234             } else bbox.ReverseSet();
235             new_bodies.Add(bbox);
236             systembox.PackStart(bbox, true, true, 3);
237         }
238         foreach (BodyBox b in new_bodies) {
239             b.ResetParents();
240         }
241         this.ShowAll();
242     }
243     protected void OnHelpClick(object obj, System.EventArgs args) {
244         OpenHTML("help.html");
245     }
246     protected void Message(String s) {
247         var window = new Window("Message");
248         var container = new VBox(homogeneous: true, spacing: 3);
249         window.Add(container);
250         container.PackStart(new Label(s), false, false, 3);

```

```

251         var closeButton = new Button("Close");
252         closeButton.Clicked += delegate { window.Destroy(); };
253         container.PackStart(closeButton, false, false, 3);
254         window.ShowAll();
255     }
256     protected void OpenHTML(String relPath) {
257         System.Threading.Tasks.Task.Run(() =>
System.Diagnostics.Process.Start(relPath));
258     }
259     public void Remove(BodyBox b) {
260         var name = b.name.Text;
261         new_bodies.Remove(b);
262         systembox.Remove(b);
263         foreach (BodyBox a in new_bodies) {
264             a.ResetParents();
265         }
266     }
267 }
268 public void OnNameChanged(object obj, EventArgs args) {
269     foreach (BodyBox b in new_bodies) {
270         b.ResetParents();
271     }
272 }
273 }
274 public class BodyBox : HBox {
275     public Body body {get; set;}
276     public Entry name {get; set;}
277     public ComboBoxText parent {get; set;} = new ComboBoxText();
278     public Scale MassScale {get; set;}
279     public Scale RadiusScale {get; set;}
280     public Scale SLRScale {get; set;}
281     public Scale EScale {get; set;}
282     public Scale IncScale {get; set;}
283     public Scale ANLScale {get; set;}
284     public Scale PAScale {get; set;}
285     public Scale TAScale {get; set;}
286     public Scale RScale {get; set;}
287     public Scale GScale {get; set;}
288     public Scale BScale {get; set;}
289     public CheckButton CenterButton {get; set;}
290     public Button DeleteButton {get; set;}
291     private static readonly double ECCENTRICITY_MAX = 3;
292     public BodyBox() {}
293     public BodyBox(Menu menu, bool homogeneous = false, int spacing =
3) : base(homogeneous, spacing) {
294         body = new Structures.Body();
295         name = new Entry();
296         name.IsEditable = true;
297         name.Changed += new EventHandler(menu.OnNameChanged);
298         ResetParents();
299         MassScale = new Scale(Orientation.Vertical, 0.1, 50, 0.01);
300         RadiusScale = new Scale(Orientation.Vertical, 0.1, 1000000, 0.1);
301         SLRScale = new Scale(Orientation.Vertical, 0.1, 50, 0.01);
302         EScale = new Scale(Orientation.Vertical,
0, ECCENTRICITY_MAX, 0.001);
303         IncScale = new Scale(Orientation.Vertical, 0, 180, 0.01);
304         ANLScale = new Scale(Orientation.Vertical, 0, 359.99, 0.01);
305         PAScale = new Scale(Orientation.Vertical, 0, 359.99, 0.01);
306         TAScale = new Scale(Orientation.Vertical, 0, 359.99, 0.01);
307         RScale = new Scale(Orientation.Horizontal, 0, 1, 0.01);
308         RScale.Value = 1;
309         GScale = new Scale(Orientation.Horizontal, 0, 1, 0.01);
310         GScale.Value = 1;
311         BScale = new Scale(Orientation.Horizontal, 0, 1, 0.01);
312         BScale.Value = 1;
313         CenterButton = new CheckButton("Focusable");

```

```

314         DeleteButton = new Button("Delete");
315         DeleteButton.Clicked += new EventHandler(OnDeleteClick);
316
317         parent.Changed += new EventHandler(OnParentChange);
318         MassScale.Inverted = true;
319         RadiusScale.Inverted = true;
320         SLRScale.Inverted = true;
321         EScale.Inverted = true;
322         IncScale.Inverted = true;
323         ANLScale.Inverted = true;
324         PAScale.Inverted = true;
325         TAScale.Inverted = true;
326         var pBox = new VBox(homogeneous: false, spacing: 3);
327         pBox.PackStart(new Label("Parent Body"), false, false, 3);
328         pBox.PackStart(parent, true, true, 3);
329         var mBox = new VBox(homogeneous: false, spacing: 3);
330         mBox.PackStart(new Label("ln(m)"), false, false, 3);
331         mBox.PackStart(MassScale, true, true, 3);
332         var rBox = new VBox(homogeneous: false, spacing: 3);
333         rBox.PackStart(new Label("r (km)"), false, false, 3);
334         rBox.PackStart(RadiusScale, true, true, 3);
335         var slrBox = new VBox(homogeneous: false, spacing: 3);
336         slrBox.PackStart(new Label("ρ (AU)"), false, false, 3);
337         slrBox.PackStart(SLRScale, true, true, 3);
338         var eBox = new VBox(homogeneous: false, spacing: 3);
339         eBox.PackStart(new Label("e"), false, false, 3);
340         eBox.PackStart(EScale, true, true, 3);
341         var incBox = new VBox(homogeneous: false, spacing: 3);
342         incBox.PackStart(new Label("i (°)"), false, false, 3);
343         incBox.PackStart(IncScale, true, true, 3);
344         var anlBox = new VBox(homogeneous: false, spacing: 3);
345         anlBox.PackStart(new Label("Ω (°)"), false, false, 3);
346         anlBox.PackStart(ANLScale, true, true, 3);
347         var paBox = new VBox(homogeneous: false, spacing: 3);
348         paBox.PackStart(new Label("ω (°)"), false, false, 3);
349         paBox.PackStart(PAScale, true, true, 3);
350         var taBox = new VBox(homogeneous: false, spacing: 3);
351         taBox.PackStart(new Label("ν (°)"), false, false, 3);
352         taBox.PackStart(TAScale, true, true, 3);
353
354         this.PackStart(name, true, true, 3);
355         this.PackStart(pBox, false, false, 3);
356         this.PackStart(mBox, true, true, 3);
357         this.PackStart(rBox, true, true, 3);
358         this.PackStart(slrBox, true, true, 3);
359         this.PackStart(eBox, true, true, 3);
360         this.PackStart(incBox, true, true, 3);
361         this.PackStart(anlBox, true, true, 3);
362         this.PackStart(paBox, true, true, 3);
363         this.PackStart(taBox, true, true, 3);
364
365         var colorbox = new VBox(homogeneous: false, spacing: 3);
366         colorbox.PackStart(new Label("RGB"), false, false, 3);
367         colorbox.PackStart(RScale, true, true, 3);
368         colorbox.PackStart(GScale, true, true, 3);
369         colorbox.PackStart(BScale, true, true, 3);
370
371         this.PackStart(colorbox, true, true, 3);
372         var optionsbox = new VBox(homogeneous: false, spacing: 3);
373         optionsbox.PackStart(DeleteButton, true, true, 3);
374         optionsbox.PackStart(DeleteButton, true, true, 3);
375         this.PackStart(optionsbox, true, true, 3);
376
377     }
378     protected void OnParentChange(object obj, EventArgs args) {
379         try {

```



```

371         var parentBody = menu.new_bodies.FirstOrDefault(b =>
    b.body.name == parent.ActiveText).body;
372         double hillrad = parentBody.HillRadius()/AU;
373         this.SLRScale.Digits = Math.Max(0,8);//3-(int)Math.Log
(hillrad/100000));
374         this.SLRScale.SetIncrements(Math.Pow(10,-
this.SLRScale.Digits),hillrad/100000);
375         this.SLRScale.SetRange(Math.Pow(10,-
this.SLRScale.Digits),hillrad);
376     } catch (NullReferenceException) {} // no parent, don't set values
377 }
378     protected void OnDeleteClick(object obj, EventArgs args) {
379         menu.Remove(this);
380         menu.ShowAll();
381         this.Destroy();
382     }
383     public void Set() {
384         if (parent.ActiveText != this.name.Text && parent.Active != -1) {
385             var elements = new Structures.OrbitalElements() {
386                 semilatusrectum = SLRScale.Value*AU,
387                 eccentricity = EScale.Value,
388                 inclination = IncScale.Value*deg,
389                 ascendingNodeLongitude = ANLScale.Value*deg,
390                 periapsisArgument = PAScale.Value*deg,
391                 trueAnomaly = TAScale.Value*deg
392             };
393             body = new Structures.Body(menu.new_bodies.FirstOrDefault(b
=> b.body.name == parent.ActiveText).body,elements);
394         }
395         body.name = this.name.Text;
396         body.stdGrav = Math.Pow(Math.E,MassScale.Value)*G*1e22;
397         body.radius = RadiusScale.Value*1e3;
398         body.color = new Vector3(RScale.Value, GScale.Value,
BScale.Value);
399     }
400     public void SetElements(OrbitalElements elements) {
401         try {
402             if (elements.semilatusrectum > this.body.parent.HillRadius()/
AU) {
403                 SLRScale.SetRange(1e-8,elements.semilatusrectum);
404             }
405             } catch (NullReferenceException) {} // body has no parent, we
cannot check the slr
406         SLRScale.Value = elements.semilatusrectum/AU;
407         if (elements.eccentricity > ECCENTRICITY_MAX) {
408             EScale.SetRange(0,elements.eccentricity);
409         } else {
410             EScale.SetRange(0, ECCENTRICITY_MAX); // there is no way to
see the current range, so we'll set it every time
411         }
412         EScale.Value = elements.eccentricity;
413         IncScale.Value = elements.inclination/deg;
414         ANLScale.Value = elements.ascendingNodeLongitude/deg;
415         PAScale.Value = elements.periapsisArgument/deg;
416         TAScale.Value = elements.trueAnomaly/deg;
417     }
418     public void ReverseSet(bool elem = true) {
419         if (elem) try {
420             parent.Active = menu.new_bodies.FindIndex(b => b.name.Text
== body.parent.name);
421             var elements = new OrbitalElements(body.position-
body.parent.position,body.velocity-body.parent.velocity,body.parent.stdGrav);
422             this.SetElements(elements);
423         } catch (NullReferenceException) {} // if body has no parent
424         name.Text = body.name;
425         MassScale.Value = Math.Log((body.stdGrav/G)/1e22);

```



```
426         RadiusScale.Value = body.radius/1e3;
427         RScale.Value = body.color.x;
428         GScale.Value = body.color.y;
429         BScale.Value = body.color.z;
430     }
431     public void ResetParents() {
432         parent.RemoveAll();
433         foreach (BodyBox b in menu.new_bodies) {
434             parent.AppendText(b.name.Text);
435         }
436         try {
437             parent.Active = menu.new_bodies.FindIndex(b => b.name.Text ==
body.parent.name);
438         } catch (NullReferenceException) {} // parent no longer exists
439     }
440 }
441 }
442 [Serializable()]
443 public class SaveData {
444     public List<Body> bodies {get; set;}
445     public List<OrbitalElements> elements {get; set;}
446     public List<bool> centers {get; set;}
447     public double timestep {get; set;}
448     public double radius_multiplier {get; set;}
449     public double line_max {get; set;}
450 }
451 }
```

```

1  # Help
2
3  Main variables:
4
5  Name | Description
6  -----|-----
7  Mechanics Timestep | The time in seconds that acceleration is assumed
   to be constant for. Set higher for faster simulation, lower for more accurate
   simulation
8  Planetary Radii Multiplier | The factor by which the size of bodies are
   multiplied when drawn to the screen. (At real size, most planets cannot be
   seen)
9  Orbit Trail Length | The length of the trails of each body, as a
   number of timesteps
10
11 Below are descriptions of the various orbital elements you can change
12
13 Name | Description | Symbol | Range
14 -----|-----|-----|-----
15 Semi-latus rectum | the distance between two bodies at right
   angles to the "periapsis" (minimum point) |  $p$  |  $[0,+\infty)$ 
16 Eccentricity | A measure of the shape of the orbit,
   illustrated below |  $e$  |  $[0,+\infty)$ 
17 Inclination | the angle between the orbital plane and the
   reference plane |  $i$  |  $\backslash[0,180\backslash]$ 
18 Longitude of the ascending node | the angle from the reference direction
   anticlockwise to the point where the orbiting body rises above the reference
   plane |  $\Omega$  |  $[0,360)$ 
19 Argument of periapsis | the angle from the ascending node
   anticlockwise to the periapsis. |  $\omega$  |  $[0,360)$ 
20 True anomaly | the angle from the periapsis anticlockwise
   to the current position of the body. |  $v$  |  $[0,360)$ 
21
22 The best way to understand how these work is to modify the variables of an
   existing system. For each body you can also edit its name and which planet it
   is orbiting. If it is set to not be orbiting any planet, the orbital elements
   will be ignored and it will be placed at the origin with 0 velocity.
23
24 ### Eccentricity/Semi-latus rectum:
25
26 ![eccentricity](help/eccentricity.jpg "Eccentricity")
27
28 ### Existing Systems
29
30 There are several existing systems to try:
31
32 Name | Description
33 -----|-----
34 Standard | Our solar system
35 Inner | The inner 4 planets of our solar system
36 EccentricityDemo | A demonstration of how eccentricity affects
   orbits, as shown above
37 RoguePlanet1/RoguePlanet2 | Two examples of the effects of a rogue planet
   entering our solar system
38 SuperJupiterEarth | An example of three body mechanics, with earth
   orbiting a planet much more massive than jupiter
39 Binary | A binary star system, showing non-Keplerian
   orbital mechanics
40
41 ### In Simulation Controls:
42
43 Control | Effect
44 -----|-----
45 Esc | Pause and edit variables
46 L | camera lock
47 R | Reset camera

```

---

48	F		Change camera focus
49	C		Toggle stereoscopic camera
50	P		Pause
51	Mouse		Move Camera
52	Scroll		Zoom
53	Up/Down		Increase/Decrease planetary radii multiplier
54	Right/Left		Increase/Decrease mechanics timestep
55	PgUp/PgDown		Increase/Decrease orbit trail length

# Help

Main variables:

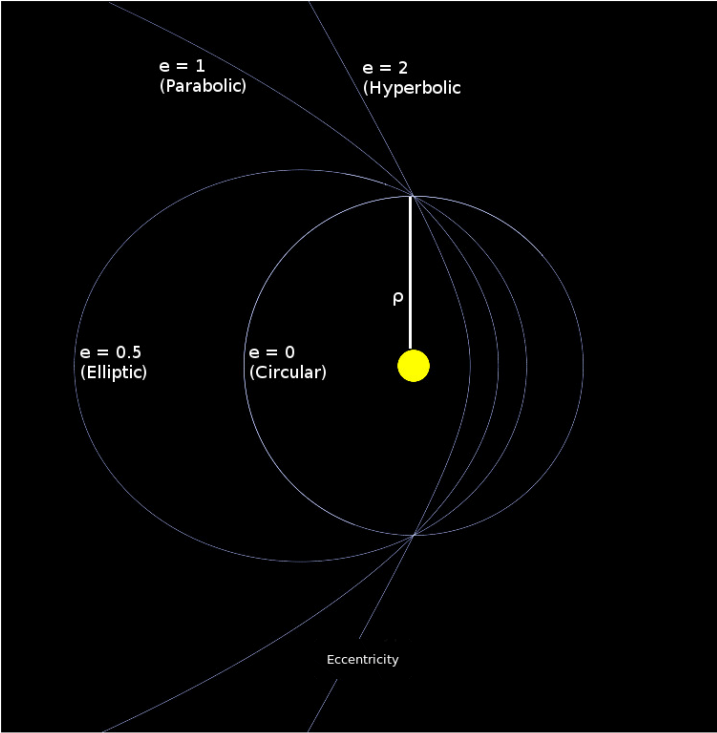
Name	Description
Mechanics Timestep	The time in seconds that acceleration is assumed to be constant for. Set higher for faster simulation, lower for more accurate simulation
Planetary Radii Multiplier	The factor by which the size of bodies are multiplied when drawn to the screen. (At real size, most planets cannot be seen)
Orbit Trail Length	The length of the trails of each body, as a number of timesteps

Below are descriptions of the various orbital elements you can change

Name	Description	Symbol	Range
Semi-latus rectum	the distance between two bodies at right angles to the "periapsis" (minimum point)	$\rho$	$[0, +\infty)$
Eccentricity	A measure of the shape of the orbit, illustrated below	$e$	$[0, +\infty)$
Inclination	the angle between the orbital plane and the reference plane	$i$	$[0, 180]$
Longitude of the ascending node	the angle from the reference direction anticlockwise to the point where the orbiting body rises above the reference plane	$\Omega$	$[0, 360)$
Argument of periapsis	the angle from the ascending node anticlockwise to the periapsis.	$\omega$	$[0, 360)$
True anomaly	the angle from the periapsis anticlockwise to the current position of the body.	$\nu$	$[0, 360)$

The best way to understand how these work is to modify the variables of an existing system. For each body you can also edit its name and which planet it is orbiting. If it is set to not be orbiting any planet, the orbital elements will be ignored and it will be placed at the origin with 0 velocity.

Eccentricity/Semi-latus rectum:



## Existing Systems

There are several existing systems to try:

Name	Description
Standard	Our solar system
Inner	The inner 4 planets of our solar system
EccentricityDemo	A demonstration of how eccentricity affects orbits, as shown above
RoguePlanet1/RoguePlanet2	Two examples of the effects of a rogue planet entering our solar system
SuperJupiterEarth	An example of three body mechanics, with earth orbiting a planet much more massive than jupiter
Binary	A binary star system, showing non-Keplerian orbital mechanics

In Simulation Controls:

Control	Effect
Esc	Pause and edit variables
L	camera lock
R	Reset camera
F	Change camera focus
C	Toggle stereoscopic camera
P	Pause
Mouse	Move Camera
Scroll	Zoom
Up/Down	Increase/Decrease planetary radii multiplier
Right/Left	Increase/Decrease mechanics timestep
PgUp/PgDown	Increase/Decrease orbit trail length

```

1  <?xml version="1.0"?>
2  <SaveData xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
   www.w3.org/2001/XMLSchema-instance">
3    <bodies>
4      <Body>
5        <name>Sol</name>
6        <stdGrav>1.3271244001799964E+20</stdGrav>
7        <radius>139268400</radius>
8        <position>
9          <x>0</x>
10         <y>0</y>
11         <z>0</z>
12       </position>
13       <velocity>
14         <x>0</x>
15         <y>0</y>
16         <z>0</z>
17       </velocity>
18       <color>
19         <x>1</x>
20         <y>1</y>
21         <z>0</z>
22       </color>
23     </Body>
24     <Body>
25       <name>Mercury</name>
26       <parent>
27         <name>Sol</name>
28         <stdGrav>1.3271244001799964E+20</stdGrav>
29         <radius>139268400</radius>
30         <position>
31           <x>0</x>
32           <y>0</y>
33           <z>0</z>
34         </position>
35         <velocity>
36           <x>0</x>
37           <y>0</y>
38           <z>0</z>
39         </velocity>
40         <color>
41           <x>1</x>
42           <y>1</y>
43           <z>0</z>
44         </color>
45       </parent>
46       <stdGrav>22032999999999.996</stdGrav>
47       <radius>2439700</radius>
48       <position>
49         <x>56094857268.929779</x>
50         <y>18470808038.685459</y>
51         <z>-3639860019.8482571</z>
52       </position>
53       <velocity>
54         <x>-23059.91143315248</x>
55         <y>40410.16718732578</y>
56         <z>5417.5320731151887</z>
57       </velocity>
58       <color>
59         <x>0.56046296135775409</x>
60         <y>0.55068107762906127</y>
61         <z>0.56157095509448862</z>
62       </color>
63     </Body>
64     <Body>
65       <name>Venus</name>

```

```

66     <parent>
67         <name>Sol</name>
68         <stdGrav>1.3271244001799964E+20</stdGrav>
69         <radius>139268400</radius>
70         <position>
71             <x>0</x>
72             <y>0</y>
73             <z>0</z>
74         </position>
75         <velocity>
76             <x>0</x>
77             <y>0</y>
78             <z>0</z>
79         </velocity>
80         <color>
81             <x>1</x>
82             <y>1</y>
83             <z>0</z>
84         </color>
85     </parent>
86     <stdGrav>324860000000000</stdGrav>
87     <radius>6051800</radius>
88     <position>
89         <x>94026719953.375473</x>
90         <y>54820791927.505112</y>
91         <z>-4678317190.0343094</z>
92     </position>
93     <velocity>
94         <x>-17459.958764413113</x>
95         <y>30051.484046928446</y>
96         <z>1418.5080376387518</z>
97     </velocity>
98     <color>
99         <x>0.72900576136582407</x>
100        <y>0.71637682452381213</y>
101        <z>0.67915792131715791</z>
102    </color>
103 </Body>
104 <Body>
105     <name>Earth</name>
106     <parent>
107         <name>Sol</name>
108         <stdGrav>1.3271244001799964E+20</stdGrav>
109         <radius>139268400</radius>
110         <position>
111             <x>0</x>
112             <y>0</y>
113             <z>0</z>
114         </position>
115         <velocity>
116             <x>0</x>
117             <y>0</y>
118             <z>0</z>
119         </velocity>
120         <color>
121             <x>1</x>
122             <y>1</y>
123             <z>0</z>
124         </color>
125     </parent>
126     <stdGrav>398600441899999.75</stdGrav>
127     <radius>6371000</radius>
128     <position>
129         <x>-137661649561.86435</x>
130         <y>-59604436913.903717</y>
131         <z>-52017.055211908657</z>

```

```
132     </position>
133     <velocity>
134         <x>11350.961632140883</x>
135         <y>-27447.999679624696</y>
136         <z>-0.023953990486544459</z>
137     </velocity>
138     <color>
139         <x>0.36141510867913057</x>
140         <y>0.3805593555251558</y>
141         <z>0.46848657909765851</z>
142     </color>
143 </Body>
144 <Body>
145     <name>Mars</name>
146     <parent>
147         <name>Sol</name>
148         <stdGrav>1.3271244001799964E+20</stdGrav>
149         <radius>139268400</radius>
150         <position>
151             <x>0</x>
152             <y>0</y>
153             <z>0</z>
154         </position>
155         <velocity>
156             <x>0</x>
157             <y>0</y>
158             <z>0</z>
159         </velocity>
160         <color>
161             <x>1</x>
162             <y>1</y>
163             <z>0</z>
164         </color>
165     </parent>
166     <stdGrav>42828370000000</stdGrav>
167     <radius>3389500</radius>
168     <position>
169         <x>192837866664.45142</x>
170         <y>74352850962.384216</y>
171         <z>-3185678152.8402276</z>
172     </position>
173     <velocity>
174         <x>-9683.53452530936</x>
175         <y>24648.163414061142</y>
176         <z>754.57965242323269</z>
177     </velocity>
178     <color>
179         <x>0.5128845217545257</x>
180         <y>0.33674146859646792</y>
181         <z>0.20228389324126941</z>
182     </color>
183 </Body>
184 <Body>
185     <name>Jupiter</name>
186     <parent>
187         <name>Sol</name>
188         <stdGrav>1.3271244001799964E+20</stdGrav>
189         <radius>139268400</radius>
190         <position>
191             <x>0</x>
192             <y>0</y>
193             <z>0</z>
194         </position>
195         <velocity>
196             <x>0</x>
197             <y>0</y>
```

```

198         <z>0</z>
199     </velocity>
200     <color>
201         <x>1</x>
202         <y>1</y>
203         <z>0</z>
204     </color>
205 </parent>
206 <stdGrav>1.266865349999999E+17</stdGrav>
207 <radius>69911000</radius>
208 <position>
209     <x>-644710440256.616</x>
210     <y>376632359073.35754</y>
211     <z>12869390562.507067</z>
212 </position>
213 <velocity>
214     <x>-7162.1797854799461</x>
215     <y>-11558.450238815309</y>
216     <z>208.68221379818667</z>
217 </velocity>
218 <color>
219     <x>0.71895966676826173</x>
220     <y>0.6638891549711422</y>
221     <z>0.63619163727667227</z>
222 </color>
223 </Body>
224 <Body>
225     <name>Saturn</name>
226     <parent>
227         <name>Sol</name>
228         <stdGrav>1.3271244001799964E+20</stdGrav>
229         <radius>139268400</radius>
230         <position>
231             <x>0</x>
232             <y>0</y>
233             <z>0</z>
234         </position>
235         <velocity>
236             <x>0</x>
237             <y>0</y>
238             <z>0</z>
239         </velocity>
240         <color>
241             <x>1</x>
242             <y>1</y>
243             <z>0</z>
244         </color>
245     </parent>
246     <stdGrav>37931187999999960</stdGrav>
247     <radius>58232000</radius>
248     <position>
249         <x>-329707227630.69629</x>
250         <y>-1334017258739.801</y>
251         <z>36377148113.9408</z>
252     </position>
253     <velocity>
254         <x>9599.4900270457638</x>
255         <y>-2794.1459237464405</y>
256         <z>-332.58296276534276</z>
257     </velocity>
258     <color>
259         <x>0.82463722535772355</x>
260         <y>0.74701936767707955</y>
261         <z>0.59518943574319</z>
262     </color>
263 </Body>

```



```
264     <Body>
265       <name>Uranus</name>
266       <parent>
267         <name>Sol</name>
268         <stdGrav>1.3271244001799964E+20</stdGrav>
269         <radius>139268400</radius>
270         <position>
271           <x>0</x>
272           <y>0</y>
273           <z>0</z>
274         </position>
275         <velocity>
276           <x>0</x>
277           <y>0</y>
278           <z>0</z>
279         </velocity>
280         <color>
281           <x>1</x>
282           <y>1</y>
283           <z>0</z>
284         </color>
285       </parent>
286       <stdGrav>5793939999999998</stdGrav>
287       <radius>25362000</radius>
288       <position>
289         <x>-2632465495055.2275</x>
290         <y>-877169734076.14636</y>
291         <z>30838609991.519142</z>
292       </position>
293       <velocity>
294         <x>2442.5014002623393</x>
295         <y>-6592.1468035332646</y>
296         <z>-55.65997653243997</z>
297       </velocity>
298       <color>
299         <x>0.565224110171928</x>
300         <y>0.73594589155310219</y>
301         <z>0.8092590995342418</z>
302       </color>
303     </Body>
304     <Body>
305       <name>Neptune</name>
306       <parent>
307         <name>Sol</name>
308         <stdGrav>1.3271244001799964E+20</stdGrav>
309         <radius>139268400</radius>
310         <position>
311           <x>0</x>
312           <y>0</y>
313           <z>0</z>
314         </position>
315         <velocity>
316           <x>0</x>
317           <y>0</y>
318           <z>0</z>
319         </velocity>
320         <color>
321           <x>1</x>
322           <y>1</y>
323           <z>0</z>
324         </color>
325       </parent>
326       <stdGrav>6836529999999991</stdGrav>
327       <radius>24622000</radius>
328       <position>
329         <x>-2343820495423.8027</x>
```

```

330         <y>3813134471805.7769</y>
331         <z>-24348583510.769043</z>
332     </position>
333     <velocity>
334         <x>-4628.5890011499869</x>
335         <y>-2888.9049317875642</y>
336         <z>166.0927959831169</z>
337     </velocity>
338     <color>
339         <x>0.55252447046234221</x>
340         <y>0.73838668051490264</y>
341         <z>0.868736820570925</z>
342     </color>
343 </Body>
344 <Body>
345     <name>Pluto</name>
346     <parent>
347         <name>Sol</name>
348         <stdGrav>1.3271244001799964E+20</stdGrav>
349         <radius>139268400</radius>
350         <position>
351             <x>0</x>
352             <y>0</y>
353             <z>0</z>
354         </position>
355         <velocity>
356             <x>0</x>
357             <y>0</y>
358             <z>0</z>
359         </velocity>
360         <color>
361             <x>1</x>
362             <y>1</y>
363             <z>0</z>
364         </color>
365     </parent>
366     <stdGrav>871999999999.99988</stdGrav>
367     <radius>1186000</radius>
368     <position>
369         <x>-5388867135111.2559</x>
370         <y>-3618610906335.0234</y>
371         <z>1946143855173.7239</z>
372     </position>
373     <velocity>
374         <x>3032.8408030489541</x>
375         <y>-2928.0664031931192</y>
376         <z>-563.94405501357687</z>
377     </velocity>
378     <color>
379         <x>0.732870760490961</x>
380         <y>0.6071190239708979</y>
381         <z>0.49887046260522128</z>
382     </color>
383 </Body>
384 </bodies>
385 <elements>
386     <OrbitalElements>
387         <semilatusrectum>14959787070</semilatusrectum>
388         <eccentricity>0</eccentricity>
389         <inclination>0</inclination>
390         <ascendingNodeLongitude>0</ascendingNodeLongitude>
391         <periapsisArgument>0</periapsisArgument>
392         <trueAnomaly>0</trueAnomaly>
393     </OrbitalElements>
394     <OrbitalElements>
395         <semilatusrectum>55460545213.260788</semilatusrectum>

```

```
396     <eccentricity>0.20563068999999981</eccentricity>
397     <inclination>0.12225804517417412</inclination>
398     <ascendingNodeLongitude>0.84354677448736781</ascendingNodeLongitude>
399     <periapsisArgument>1.3518700794063605</periapsisArgument>
400     <trueAnomaly>4.4026076989214138</trueAnomaly>
401 </OrbitalElements>
402 <OrbitalElements>
403     <semilatusrectum>108203961250.7718</semilatusrectum>
404     <eccentricity>0.0067732299999999331</eccentricity>
405     <inclination>0.059248866650378151</inclination>
406     <ascendingNodeLongitude>1.3383305132010905</ascendingNodeLongitude>
407     <periapsisArgument>2.2956835759598584</periapsisArgument>
408     <trueAnomaly>3.1761454603902921</trueAnomaly>
409 </OrbitalElements>
410 <OrbitalElements>
411     <semilatusrectum>149556114720.45197</semilatusrectum>
412     <eccentricity>0.0167102200000000088</eccentricity>
413     <inclination>8.7270441438834592E-07</inclination>
414     <ascendingNodeLongitude>0</ascendingNodeLongitude>
415     <periapsisArgument>1.79676742117616</periapsisArgument>
416     <trueAnomaly>1.7534336883759865</trueAnomaly>
417 </OrbitalElements>
418 <OrbitalElements>
419     <semilatusrectum>225947693282.32028</semilatusrectum>
420     <eccentricity>0.093412329999999641</eccentricity>
421     <inclination>0.03229923767033327</inclination>
422     <ascendingNodeLongitude>0.86530876133170953</ascendingNodeLongitude>
423     <periapsisArgument>5.8650190791674612</periapsisArgument>
424     <trueAnomaly>6.20383077114501</trueAnomaly>
425 </OrbitalElements>
426 <OrbitalElements>
427     <semilatusrectum>776589102926.84888</semilatusrectum>
428     <eccentricity>0.048392659999999872</eccentricity>
429     <inclination>0.022781782726280068</inclination>
430     <ascendingNodeLongitude>1.7550359006292964</ascendingNodeLongitude>
431     <periapsisArgument>0.25750325984536626</periapsisArgument>
432     <trueAnomaly>0.60046970810728639</trueAnomaly>
433 </OrbitalElements>
434 <OrbitalElements>
435     <semilatusrectum>1422541843522.9233</semilatusrectum>
436     <eccentricity>0.054150600000000118</eccentricity>
437     <inclination>0.043362007134098136</inclination>
438     <ascendingNodeLongitude>1.9847018570370527</ascendingNodeLongitude>
439     <periapsisArgument>1.6132416870058457</periapsisArgument>
440     <trueAnomaly>0.871692826669656</trueAnomaly>
441 </OrbitalElements>
442 <OrbitalElements>
443     <semilatusrectum>2864584901454.3853</semilatusrectum>
444     <eccentricity>0.047167710000000369</eccentricity>
445     <inclination>0.013436591779400342</inclination>
446     <ascendingNodeLongitude>1.2955558093602884</ascendingNodeLongitude>
447     <periapsisArgument>2.9838888911697836</periapsisArgument>
448     <trueAnomaly>5.4669328641995376</trueAnomaly>
449 </OrbitalElements>
450 <OrbitalElements>
451     <semilatusrectum>4497921312318.0762</semilatusrectum>
452     <eccentricity>0.008585870000000136</eccentricity>
453     <inclination>0.030877841527506698</inclination>
454     <ascendingNodeLongitude>2.2989771867912894</ascendingNodeLongitude>
455     <periapsisArgument>0.78489812656674551</periapsisArgument>
456     <trueAnomaly>5.3211603470790934</trueAnomaly>
457 </OrbitalElements>
458 <OrbitalElements>
459     <semilatusrectum>5906376272436.3623</semilatusrectum>
460     <eccentricity>0.24880766000000018</eccentricity>
461     <inclination>0.29917997705373817</inclination>
```

```
462     <ascendingNodeLongitude>1.9251587278747897</ascendingNodeLongitude>
463     <periapsisArgument>3.9107027062759294</periapsisArgument>
464     <trueAnomaly>4.1700944123719523</trueAnomaly>
465   </OrbitalElements>
466 </elements>
467 <centers>
468   <boolean>>false</boolean>
469   <boolean>>false</boolean>
470   <boolean>>false</boolean>
471   <boolean>>false</boolean>
472   <boolean>>false</boolean>
473   <boolean>>false</boolean>
474   <boolean>>false</boolean>
475   <boolean>>false</boolean>
476   <boolean>>false</boolean>
477   <boolean>>false</boolean>
478 </centers>
479 <timestep>50</timestep>
480 <radius_multiplier>100</radius_multiplier>
481 <line_max>100</line_max>
482 </SaveData>
```

```

1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4  using System.Threading;
5  using System.Threading.Tasks;
6  using System.IO;
7  using Structures;
8  using static Program.Constants;
9  using Gtk;
10 using Gdk;
11 using Cairo;
12 using Graphics;
13 using static Program.Program;
14 namespace Program {
15     static class Input {
16         private static bool canMove = false;
17         private static Vector3 rootPos = null;
18         private static Vector3 rootAngle = null;
19         private static readonly double MOUSE_SENSITIVITY = 1;
20         private static readonly double SCROLL_SENSITIVITY = 1.1;
21         private static readonly double TIME_SENSITIVITY = 1.2;
22         private static readonly double RADIUS_SENSITIVITY = 1.1;
23         private static readonly int LINE_SENSITIVITY = 5;
24         private static double focal_length = -1;
25         [GLib.ConnectBefore]
26         public static void OnKeyPress(object sender, KeyPressEventArgs args) {
27             if (args.Event.Key == Gdk.Key.f) {
28                 if (Program.activesys == null) return;
29                 else {
30                     Program.activesys.IterateCenter();
31                     Program.sys_view.ClearPaths();
32                 }
33                 args.RetVal = true;
34             } else if (args.Event.Key == Gdk.Key.r) {
35                 double d = Vector3.Magnitude
36 (Program.sys_view.camera.position);
37                 Program.sys_view.camera = new Camera(d, Vector3.zero);
38             } else if (args.Event.Key == Gdk.Key.l) {
39                 canMove = !canMove;
40                 if (!canMove) {
41                     rootPos = null;
42                 }
43             } else if (args.Event.Key == Gdk.Key.Up) {
44                 Program.sys_view.radius_multiplier *= RADIUS_SENSITIVITY;
45             } else if (args.Event.Key == Gdk.Key.Down) {
46                 Program.sys_view.radius_multiplier /= RADIUS_SENSITIVITY;
47             } else if (args.Event.Key == Gdk.Key.Right) {
48                 Program.activesys.Stop();
49                 Program.timestep *= TIME_SENSITIVITY;
50                 Program.activesys.StartAsync(step: Program.timestep);
51             } else if (args.Event.Key == Gdk.Key.Left) {
52                 Program.activesys.Stop();
53                 Program.timestep /= TIME_SENSITIVITY;
54                 Program.activesys.StartAsync(step: Program.timestep);
55             } else if (args.Event.Key == Gdk.Key.Page_Down) {
56                 // don't make it smaller than 0
57                 if (Program.sys_view.line_max >= LINE_SENSITIVITY) {
58                     Program.sys_view.line_max -= LINE_SENSITIVITY;
59                 }
60             } else if (args.Event.Key == Gdk.Key.Page_Up) {
61                 Program.sys_view.line_max += LINE_SENSITIVITY;
62             } else if (args.Event.Key == Gdk.Key.Escape) {
63                 Program.sys_view.Stop();
64                 Program.activesys.Stop();
65                 Program.mainWindow.Destroy();

```

```

66         var menu = new UI.Menu();
67         var data = new UI.SaveData() {
68             bodies = ((IEnumerable<Body>)Program.activesys).ToList(),
69             centers = Program.CustomCenters,
70             timestep = Program.timestep,
71             radius_multiplier = Program.sys_view.radius_multiplier,
72             line_max = Program.sys_view.line_max
73         };
74         menu.temp_savedata = data;
75         menu.loadButton.Click();
76     } else if (args.Event.Key == Gdk.Key.q) {
77         Program.sys_view.camera = new Camera(Vector3.Magnitude
78 (Program.sys_view.camera.position)*SCROLL_SENSITIVITY,Program.sys_view.camera.angle);
79     } else if (args.Event.Key == Gdk.Key.w) {
80         Program.sys_view.camera = new Camera(Vector3.Magnitude
81 (Program.sys_view.camera.position)/
82 SCROLL_SENSITIVITY,Program.sys_view.camera.angle);
83     } else if (args.Event.Key == Gdk.Key.c) {
84         if (focal_length == -1) {
85             Console.WriteLine("hi");
86             focal_length = Vector3.Magnitude
87 (Program.sys_view.camera.position);
88         Program.sys_view.camera = new Camera
89 (1000*AU,Program.sys_view.camera.angle);
90         //Program.sys_view.ClearPaths();
91         //Program.sys_view.Redraw();
92     } else {
93         Console.WriteLine("hi2");
94         Program.sys_view.camera = new Camera
95 (focal_length,Program.sys_view.camera.angle);
96         //Program.sys_view.Redraw();
97         focal_length = -1;
98     }
99 }
100 }
101 [GLib.ConnectBefore]
102 public static void OnMouseMovement(Object sender,
103 MotionNotifyEventArgs args) {
104     if (canMove) {
105         if (rootPos == null || rootAngle == null ) {
106             rootPos = new Vector3(args.Event.X,args.Event.Y,0);
107             rootAngle = Program.sys_view.camera.angle;
108         } else {
109             double d = Vector3.Magnitude
110 (Program.sys_view.camera.position);
111             Program.sys_view.camera = new Camera(d,rootAngle +
112 deg*MOUSE_SENSITIVITY* new Vector3(rootPos.y - args.Event.Y,0,args.Event.X -
113 rootPos.x));
114         } args.RetVal = true;
115     }
116 }
117 [GLib.ConnectBefore]
118 public static void OnScrollMovement(Object sender, ScrollEventArgs
119 args) {
120     if (args.Event.Direction == Gdk.ScrollDirection.Up) {
121         Program.sys_view.bounds_multiplier /= SCROLL_SENSITIVITY;
122         Program.sys_view.camera = new Camera(Vector3.Magnitude
123 (Program.sys_view.camera.position)/
124 SCROLL_SENSITIVITY,Program.sys_view.camera.angle);
125     } else if (args.Event.Direction == Gdk.ScrollDirection.Down) {
126         Program.sys_view.bounds_multiplier *= SCROLL_SENSITIVITY;
127         Program.sys_view.camera = new Camera(Vector3.Magnitude
128 (Program.sys_view.camera.position)*SCROLL_SENSITIVITY,Program.sys_view.camera.angle);
129     }
130 }

```

---

```
118     }  
119 }
```

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using static Program.Constants;
5  namespace Structures {
6      public static class Tests {
7          public static bool MatrixTest() {
8              // Scalar Arithmetic
9              var i = new Matrix3(new Vector3(1,0,0),new Vector3(0,1,0),new
Vector3(0,0,1));
10             var a = new Matrix3(new Vector3(1,3,1),new Vector3(0,4,1),new
Vector3(2,-1,0));
11             if (i*i != i) {
12                 Console.WriteLine("i*i != i");
13                 return false;
14             }
15             if (i*a != a || a*i != a) {
16                 Console.WriteLine("a*i != a");
17                 return false;
18             }
19             if ((double)5 * a / (double)5 != a) {
20                 Console.WriteLine("5*a/5 != i");
21                 return false;
22             }
23             if (a + a != 2 * a) {
24                 Console.WriteLine("a + a != 2 * a");
25                 return false;
26             }
27
28             // Inverse (Also tests Determinant, Minor, Transpose_Cofactor)
29             var a_inv = new Matrix3(new Vector3(-1,1,1),new Vector3
(-2,2,1),new Vector3(8,-7,-4));
30             if (Matrix3.Inverse(a) != a_inv) {
31                 Console.WriteLine($"Matrix3.Inverse(i) == \n{Matrix3.Inverse
(a)} != \n{a_inv}");
32                 return false;
33             }
34             if (Matrix3.Inverse(Matrix3.Inverse(a)) != a) {
35                 Console.WriteLine("inv(inv(a)) != a");
36                 return false;
37             }
38             try {
39                 Matrix3.Inverse(new Matrix3
(Vector3.zero,Vector3.zero,Vector3.zero));
40                 Console.WriteLine("No Exception on Inverse of Singular
Matrix");
41             } catch (DivideByZeroException) {}
42
43             // Matrix-Matrix Multiplication (Also tests Transpose)
44             var a_sq = new Matrix3(new Vector3(3,14,4), new Vector3(2,15,4),
new Vector3(2,2,1));
45             if (a * a != a_sq) {
46                 Console.WriteLine($"a * a != a_sq");
47                 return false;
48             }
49             return true;
50         }
51         public static bool VectorTest() {
52             var a = new Vector3(2,3,6);
53             var z = Vector3.zero;
54             // Scalar Arithmetic
55             if (a + z != a || z + a != a) {
56                 Console.WriteLine("a + z != a");
57                 return false;
58             }
59             if ((double)5 * a / (double)5 != a) {

```



```

60         Console.WriteLine("5*a/5 != i");
61         return false;
62     }
63     if (a + a != 2 * a) {
64         Console.WriteLine("a + a != 2 * a");
65         return false;
66     }
67     if (-a != z - a) {
68         Console.WriteLine("-a != z - a");
69         return false;
70     }
71     if (Vector3.Magnitude(a) != 7) {
72         Console.WriteLine("Incorrect Magnitude");
73         return false;
74     }
75     if (Vector3.dot(new Vector3(1,2,0),new Vector3(-2,1,0)) != 0 ||
Vector3.dot(a,a) != 49) {
76         Console.WriteLine("incorrect dot");
77         return false;
78     }
79     if (Vector3.cross(new Vector3(3,-3,1), new Vector3(4,9,2)) != new
Vector3(-15,-2,39)) {
80         Console.WriteLine("incorrect cross");
81         return false;
82     }
83     var a_u = new Vector3((double)2/7,(double)3/7,(double)6/7);
84     if (Vector3.Unit(a) != a_u) {
85         Console.WriteLine("incorrect unit");
86         return false;
87     }
88     var exp = new Vector3(1000,0,-100);
89     try {
90         Console.WriteLine(Vector3.Unit(Vector3.zero));
91         Console.WriteLine("Unit(zero) did not throw exception");
92         return false;
93     } catch (DivideByZeroException) {
94     } catch (Exception) {
95         Console.WriteLine("Incorrect exception");
96         return false;
97     }
98     if (Vector3.PolarToCartesian(Vector3.CartesianToPolar(a)) != a) {
99         var b = Vector3.PolarToCartesian(Vector3.CartesianToPolar(a));
100         Console.WriteLine((a.x - b.x)/a.x);
101         Console.WriteLine("Cartesian-Polar conversions failed");
102         return false;
103     }
104     Vector3 c = null;
105     Vector3 d = null;
106     if (a == c || c != d) {
107         Console.WriteLine("Null checks incorrect");
108         return false;
109     }
110     return true;
111 }
112
113 public static bool BodyTest() {
114     var sun = new Body {
115         stdGrav = 1.3271440019e20,
116         radius = 6.95e8
117     };
118     var elem = new OrbitalElements() {
119         semilatusrectum = 3.2*AU,
120         eccentricity = 0.7,
121         inclination = 1.2,
122         ascendingNodeLongitude = 0.1,
123         periapsisArgument = 4.3,

```

[illegible]

```

    "v") && m == 0) {
176                                     // They are undefined, don't
    worry
177                                     continue;
178                                     }
179                                     Console.WriteLine($"Orbital element
test failed: {t.Item1}, {t.Item2}, {t.Item3}, {((t.Item2 - t.Item3)/
t.Item2)*100}%");
180                                     return false;
181                                     }
182                                     }
183                                     }
184                                     }
185                                     }
186                                     }
187                                     }
188                                     var elemx = new OrbitalElements() {
189                                         inclination = 2*Math.PI,
190                                         ascendingNodeLongitude = 7.5*Math.PI,
191                                         trueAnomaly = 27*Math.PI,
192                                         periapsisArgument = 3.75*Math.PI
193                                     };
194                                     if (
195                                         elemx.inclination > 1e-10 ||
196                                         (elemx.ascendingNodeLongitude - (1.5*Math.PI))/(1.5*Math.PI)
> 1e-10 ||
197                                         (elemx.trueAnomaly - Math.PI)/Math.PI > 1e-10 ||
198                                         (elemx.periapsisArgument-1.75*Math.PI)/(1.75*Math.PI) > 1e-10
199                                     ) {
200                                         Console.WriteLine("Implicit angle readjustment failed");
201                                         Console.WriteLine(elemx.trueAnomaly/Math.PI);
202                                     }
203                                     return true;
204                                     }
205                                     public static bool PlanetarySystemTest() {
206                                         List<Body> bodies = Structures.Examples.solar_system_bodies;
207                                         var sys = new PlanetarySystem(bodies);
208                                         if (!bodies.SequenceEqual(((IEnumerable<Body>)sys).ToList())) {
209                                             Console.WriteLine("Constructor does not add bodies");
210                                             return false;
211                                         }
212                                         var b = Structures.Examples.solar_system_bodies[3];
213                                         sys.Add(b);
214                                         if (sys[sys.Count - 1] != b) {
215                                             Console.WriteLine("Add() failed");
216                                             return false;
217                                         }
218                                         var position1 = new Vector3(2,-4,12);
219                                         sys = new PlanetarySystem(new List<Body>() {
220                                             new Body() {stdGrav = 10},
221                                             new Body() {
222                                                 stdGrav = 20,
223                                                 position = position1
224                                             }
225                                         });
226                                         if (sys.Barycenter() != 2*position1/3) {
227                                             Console.WriteLine("Barycenter 1 incorrect");
228                                             return false;
229                                         }
230                                         sys[1].stdGrav /= 2;
231                                         var position1polar = Vector3.CartesianToPolar(position1);
232                                         var position2polar = new Vector3
(position1polar.x,position1polar.y + Math.PI/3,position1polar.z);
233                                         sys.Add(new Body {
234                                             stdGrav = 10,
235                                             position = Vector3.PolarToCartesian(position2polar)

```

```
236         });
237         double distance = Math.Sqrt(3)/3;
238         Vector3 expected_barycenter_polar = new Vector3
239         (distance*positionlpolar.x,positionlpolar.y + Math.PI/6,positionlpolar.z);
240         if (sys.Barycenter() != Vector3.PolarToCartesian
241         (expected_barycenter_polar)) {
242             Console.WriteLine("Barycenter 2 incorrect");
243             return false;
244         }
245     }
246 }
```

```
1  using System;
2  using Structures;
3  using static Structures.Tests;
4
5  class Tests {
6      static void Main() {
7          if (VectorTest()) Console.WriteLine("Vector test complete");
8          else Console.WriteLine("Vector test failed");
9          if (MatrixTest()) Console.WriteLine("Matrix test complete");
10         else Console.WriteLine("Matrix test failed");
11         if (BodyTest()) Console.WriteLine("Body test complete");
12         else Console.WriteLine("Body test failed");
13         if (PlanetarySystemTest()) Console.WriteLine("Planetary system test
complete");
14         else Console.WriteLine("Planetary system test failed");
15     }
16 }
```