# Analysis

### Introduction

The solar system is an important part of the physics curriculum, and it’s something that children often find more interesting than other core physics topics such as circuits or mechanics. I think it’s important to teach the solar system because it’s fun, and because it can also help students understand the more boring topics, such as forces or energy, as forces and conservation of energy can be demonstrated with the solar system.

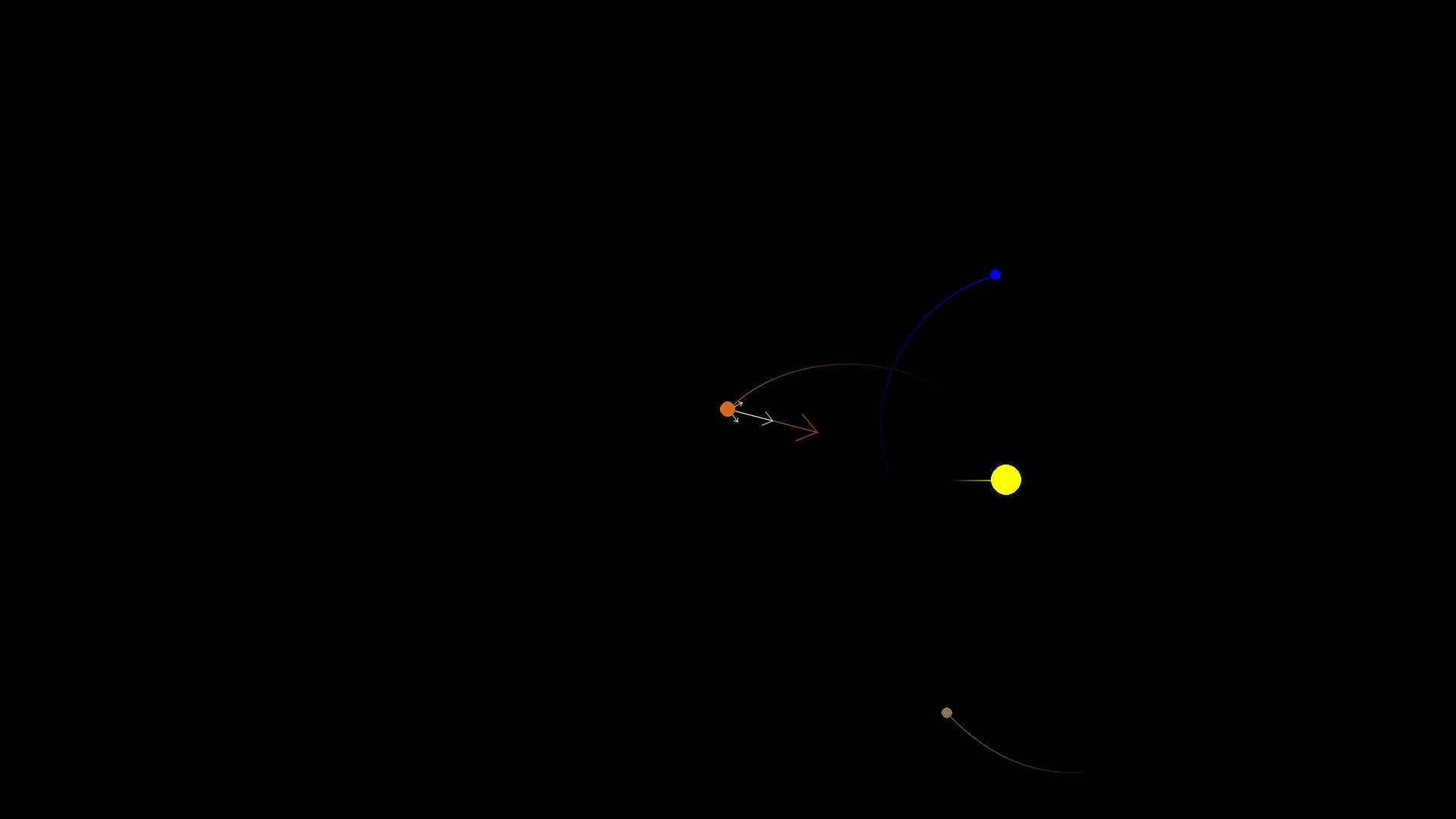
However, there is a lack of interactive tools available for teaching the solar system that also incorporate other physics topics such as energy. It would be useful to teachers, and students, to have an application that can use an interesting topic like space that also teaches important concepts from other topics such as conservation of momentum or Newton’s laws.

### How a simulation of the solar system works

The orbits of the planets are governed by two fundamental laws of physics:

* Newton’s second law of motion: , where F is the force acting on a body, m is the mass of a body, and a is the acceleration of the body
* Newton’s law of universal gravitation: once again F is the force, G is something called the “gravitation constant”, M and m represent the masses between two bodies, and r2 is the distance between them.

To demonstrate this, a force diagram is attached.



The orange planet has 4 arrows around it. These are the forces acting on it. The arrow shows the direction of each force and the length of the arrow represents the strength of the force. The white arrows are individual forces, you can see one pointing towards the brown planet, one pointing towards the blue planet, and one pointing towards the yellow sun. The arrow pointing towards the sun is much longer than the other arrows, representing that it has a stronger effect. This is because the sun has a larger mass, and so the force of gravity due to it is a lot stronger.

The brown arrow is the resultant force. This is what happens when you pull the planet in the direction of all three arrows at the same time. Because the force from the sun is so much stronger than the force due to the other planets, the resultant force is pretty similar to the force from the sun.

This simulation works fine, until planets collide. How collisions are handled will be explained later, as it’s rather technical.

### What teachers want in a simulation of the solar system

These are extracts from an interview with the South Wilts physics department

Question:

“What content is covered when teaching lower school classes?”

Answer:  
 “We do a little bit of work with Y7 looking at the order of the planets in the Solar System and the properties of some of the planets. They need to understand why we have days, seasons and years.”

“In Y11 we look at the objects in the solar system and orbital motion. They need to know that gravity provides a centripetal force and how closer planets orbit faster. Usually they get asked about orbital motion in the context of satellites – low polar vs geostationary”

How this impacts the project:

Showing the orbital motions of the planets is definitely useful, but I also need another view to show the motion of satellites around the Earth for the year 11s. This would require a 3D model of the Earth. If I included a light source in this model of the Earth, emulating the Sun, it would also help explain the days and seasons for the year 7s.

Question:

Do you currently use any interactive tools when teaching the solar system? If so, which, and how do you use them? Do you think there’s anything they’re lacking that would help kids learn better?

Answer:

Not with Y11.

Y7 have the opportunity to complete independent research on a planet in the solar system and we share this link <https://eyes.nasa.gov/apps/orrery/#/home.> It has easy links to extra information and a size comparison.

It would be good to see this combined with something that shows the orbital motion of satellites.

<https://phet.colorado.edu/sims/html/my-solar-system/latest/my-solar-system_all.html>

This simulator is great to show the effect of changing mass on the theoretical solar system. I wonder if there could be a simulator which enabled us to ‘play’ with the masses in our actual solar system to see what happens – probably this would be more useful Y11 onwards.

How this impacts the project:

I need to create something that shows the motion of planets, as well as having size comparisons, like in the link above, but also shows the motion of satellites around the Earth. Should be possible to edit the masses of objects to show the effect it has too.

Question:

What other features would you want in an interactive program that would make it useful for teaching?

Answer:

With younger students it is good to be able to direct them to suitable sources of information when setting them independent tasks to avoid them just copying and pasting things they don’t understand.

Impact on project:

Planets, satellites and other celestial objects should have links to reliable, easy to digest sources of information about them when clicked on.

Question:

What format would be most convenient, e.g. would a website be preferable to an app you had to download?

Answer:

A website option would work well.

Question:

Should the program focus entirely on the solar system, or would it be useful to also use the orbits to demonstrate concepts like Newton’s laws, conservation of energy and conservation of momentum?

Answer:

We look at the solar system right at the end of Y11 so if there was a program which linked to key Newtonian Physics, it would be a great way to re-visit these laws in a different context.

Impact on project:

Should include things such as force diagrams and total energy to link the solar system to these concepts.

A review of the features requested:

1. Links to extra information
2. Size comparison
3. Showing days, seasons, years
4. Showing satellite orbits orbits
5. Interactively changing the mass of the planets
6. Demonstrating concepts of Newtonian physics
7. Hosting on a website

Feature 1 is very simple to implement. Celestial bodies will be stored as objects. It would be very easy to add a link to a reliable source as an attribute for each object, and then display it when the object is clicked on.

Feature 2 should also be rather simple. It can be done by splitting the screen in half, displaying the chosen object in the left half, and scaling the left half so that it fits the chosen object. Then display the other chosen object for comparison in the right half at the same scale.

I will lump features 3 and 4 together, as they both require the same thing. Making the model of the solar system 3D. This is challenging, 3D graphics are difficult, particularly in python. These would be great features to have, but may be beyond the scope of the project. The best way to approach them seems to be through Panda3D.

Feature 5 is doable. The mass of planets is already stored as an attribute for them. All that needs to be done is to let users edit the mass of said planets.

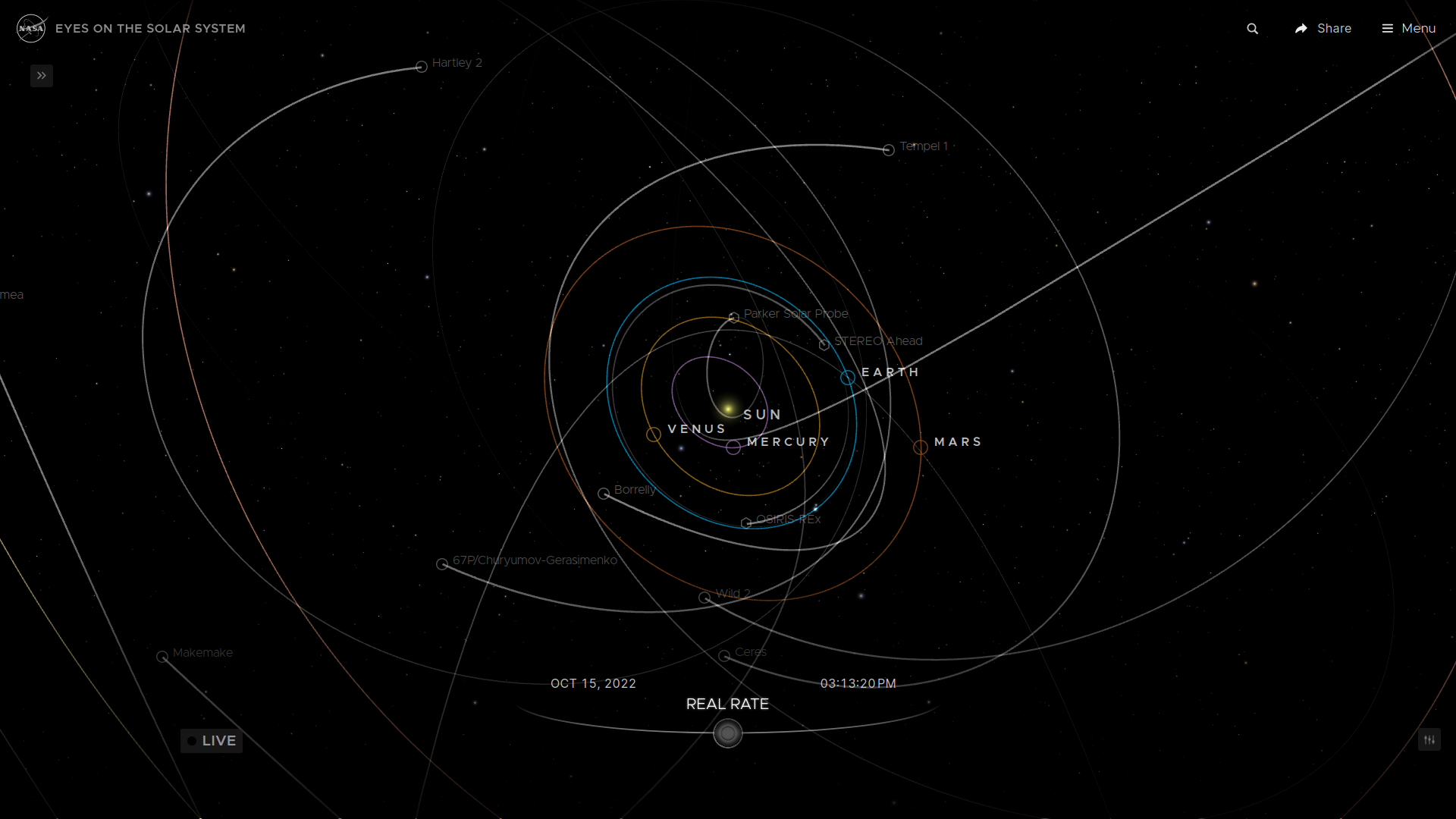
For feature 6 the challenge is figuring out how to do it in a way that a 12 year old can understand. The program already conserves momentum, energy, and uses Newton’s laws of motion because it’s supposed to be an accurate simulation. The hard bit is trying to make it clear to kids how those things are happening in the program, and help them get a better understanding of it. I will have some back and forth on the best way to do this with the client.

Feature 7 may be tricky to implement, and might have to be scrapped, but I will give it a go.

## Current alternatives

### NASA’s Eyes

<https://eyes.nasa.gov/apps/orrery/#/home>

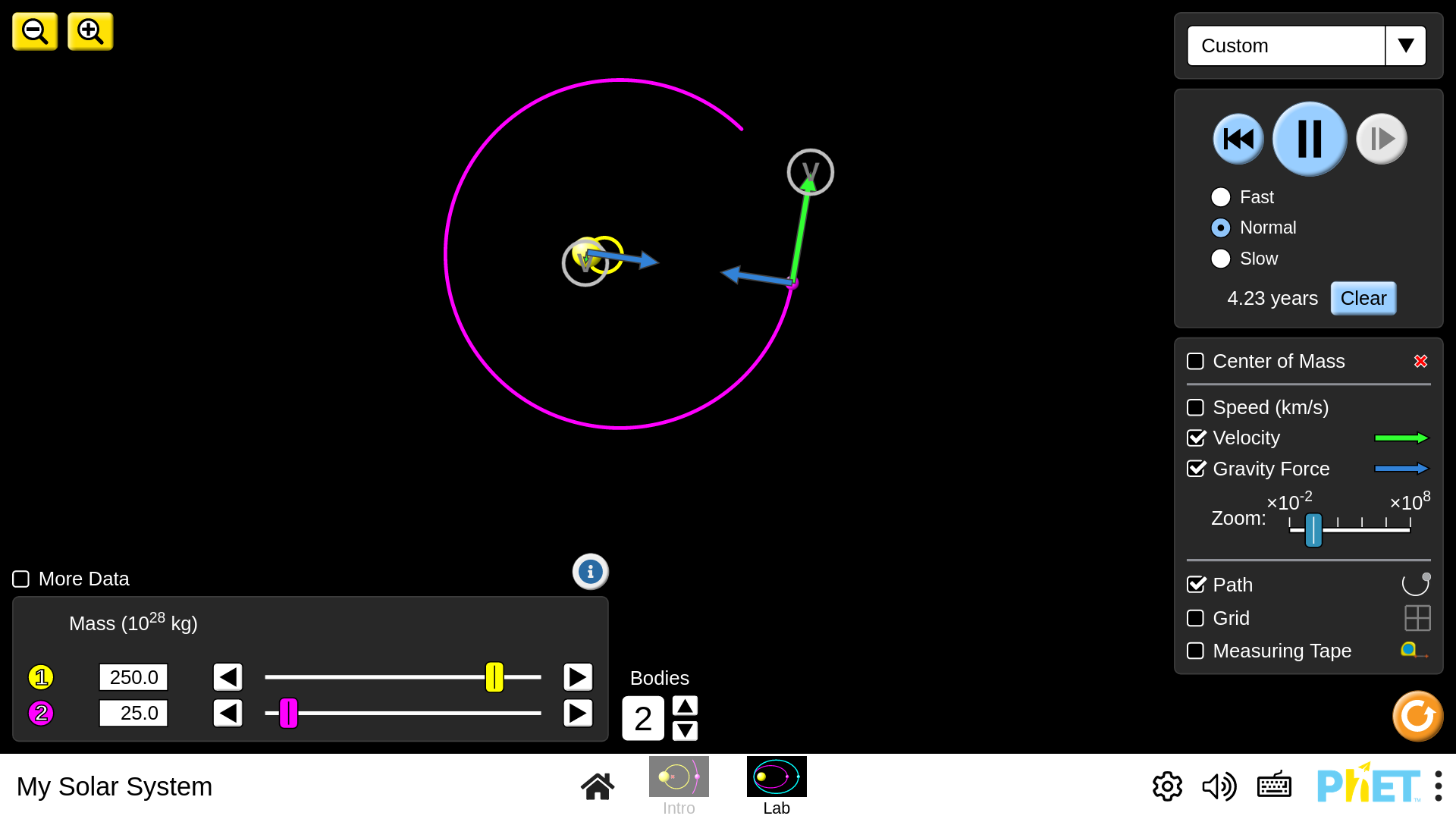


Eyes is a suite of products developed by NASA used for visualising their data. Their orrery, shown in the figure above, demonstrates the orbits of planets, asteroids, comets, and certain satellites. It is freely available from any browser, no download required, which is very convenient. My school currently uses this to teach year 7’s, primarily because it has links to further reading for their research project. It also accurately shows the trajectories of planets and other celestial bodies. Since the planets are 3d it can also be used to explain the days and seasons. This makes it a useful teaching tool.

However, the site lacks a few things. There are very few satellites shown orbiting Earth, and not enough to explain the different kind of satellite orbits like geostationary, polar etc. There is also no connection to Newtonian mechanics, nothing demonstrating it, and no way to change the mass of the planets.

### PhET: My Solar System

<https://phet.colorado.edu/sims/html/my-solar-system/latest/my-solar-system_all.html>



PhET (Physics Education Technology) is a website hosting a large amount of interactive physics simulations where students can tweak parameters and see the effects in real time. Its simulations are used by schools all around the world, and mine is no exception. As mentioned in the interview, my school finds the PhET simulation for the solar system useful because it lets you change the masses of the bodies in the solar system. It also includes force and velocity vectors, which can be used to help link back to Newtonian mechanics.

The drawback is that it’s all on a fictional solar system. You can’t put it into context with our actual planets. This would both be entertaining for students (who doesn’t want to know what would happen if the Moon weighed as much as Jupiter?) and help them visualise the effects of changing mass.

### Orbtrack

<https://www.orbtrack.org/>



Orbtrack is a tool for tracking satellites. It’s not an educational tool, but I’m including it here as showing satellite orbits was a requested feature from my physics department. Orbtrack allows you to select any satellite and show its path across a flat, 2d map. The reason for this is because showing it on a 2d map is more practical for users that want to see when satellites will pass over them.

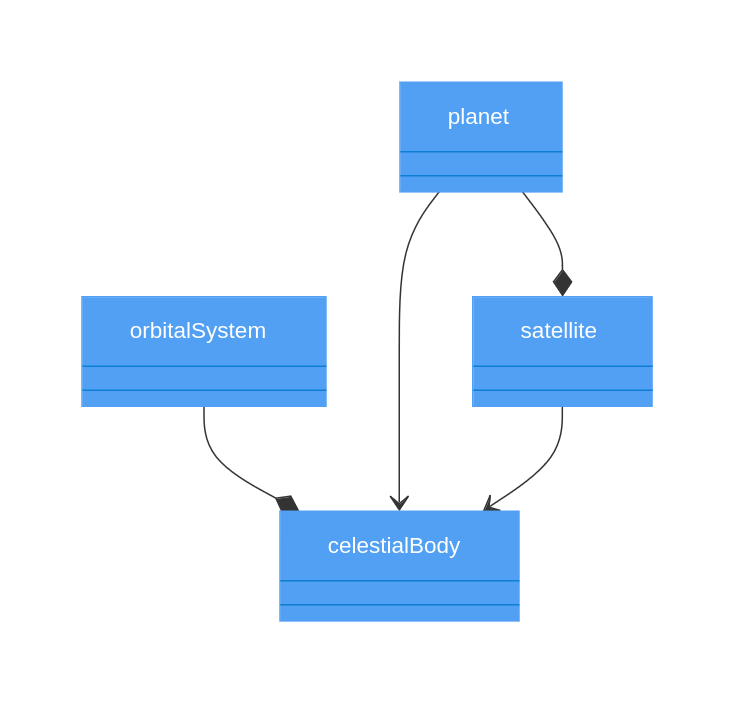
The drawbacks of this is that it’s 2d, which is not good for teaching as students will find the strange patterns made by orbital paths projected onto a 2d surface hard to understand, and it can only show one satellite at a time. I will aim to use a 3d model in my program.

## What I’ve learned from the alternatives

* A website is by far the most accessible format for teachers
* Interactivity and being able to edit parts of the system is beneficial to learning and should be in the program
* The program should show the actual solar system, not a fictional one
* Arrows and other visual tools should be used to intuitively show physical quantities such as force and velocity

## Modelling the problem

### Class diagram



An orbital system is a system of bodies orbiting each other. They are made up of these celestial bodies, and those celestial bodies are destroyed when the orbital system is destroyed, hence we use composition between them.

Planets and satellites are both celestial bodies, so they inherit from that class. Planets sometimes have satellites attached to them, and those satellites are destroyed when the planets are, so the relation between them is composition.

### NASA’s API

The client requested the ability to change masses (simple), but for real planets in our solar system (less simple). This means I need to somehow have data on all of the planets, and to keep it up to date. Luckily, NASA has a solution for us:

<https://ssd.jpl.nasa.gov/horizons/>

Horizons is an online database that hosts real time information on planets. The idea will be to access this data using their API or command-line interface, both of which use HTTP POST queries, and then translate that data to parameters for planets in my program, and load the solar system. Parsing the data may be tricky, but once I have the mass, positions and velocites of the planets it should be rather simple to load the solar system. This is because planets are already stored as objects with all of those attributes in my code, so I just need to create objects with their attributes set to be the ones retrieved from the Horizons query.

### The maths of simulating a solar system

### *Verlet Integration*

Verlet integration is a computationally efficient method for simulating the motion of objects.

This is the equation for basic Verlet:

This looks very scary, but isn’t actually that bad. is the current position of the object. is its previous position. is its updated position. is the acceleration of the object. means a very very small step forward in time.

We can do some factoring to get a more useful form of the equation:

This is the equation that will actually be used in code. What it means is that, given an object at position , and with a certain speed , its position after moving forward a small amount of time () is its old position + its speed multiplied by that small amount of time. Example:

A car is 9 miles along a road (so ). It is moving at 60 miles an hour. After 1 minute ( = 1 min = 1/60 hr), the new position of the car will be miles along the road.

So, all we need to do to find the new positions of our planets after each tick, is add their current velocities multiplied by a time factor ( to their current position.

There is also a similar equation for velocity and acceleration that will be used throughout the code:

Where you add the acceleration to get the new velocity.

### *Gravity*

As mentioned earlier, the two equations used to simulate gravity are:

The goal here is simply to work out the acceleration on a planet due to gravity. Once that is done, the acceleration of the planet is handed over to the Verlet integration algorithm, which handles everything else.

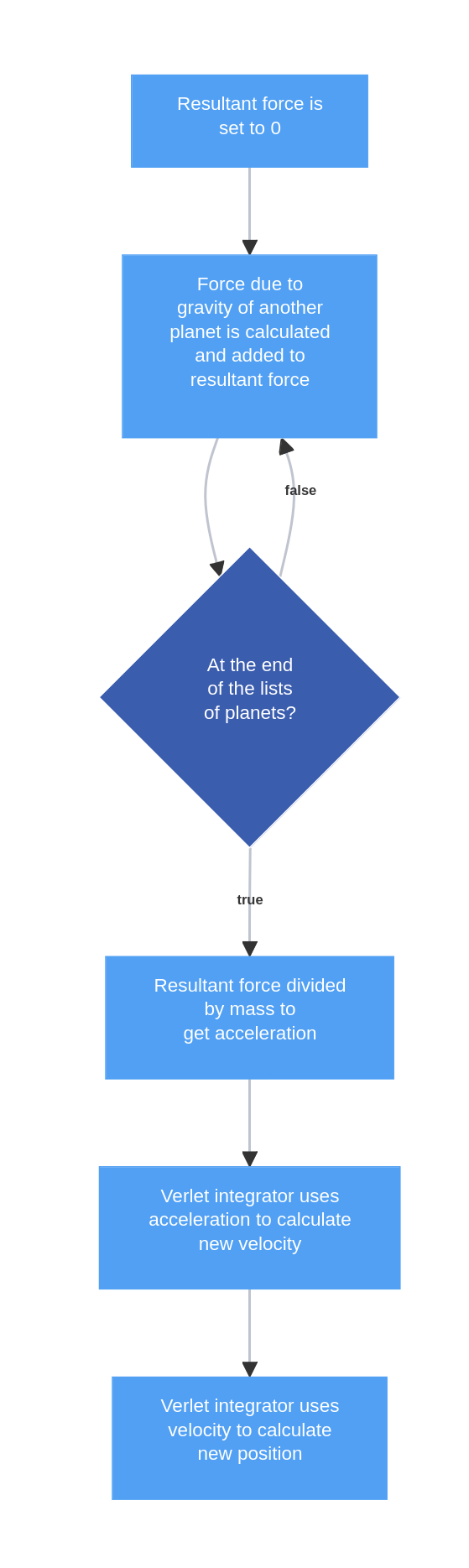
Let’s start with the simpler equation, . Here, F is the force, m is the mass of the object, and a is the acceleration. We will calculate the force using the other equation, , and we already know the mass of all of our planets. We want the acceleration. So, solving for acceleration we get:

Which is quite nice. To find the acceleration, we just take the force acting on the planet and divide it by its mass.

Now let’s cover . Here, M and m are the masses of the two objects in question (M is conventionally the mass of the larger planet), r is the distance between them and G is a physical constant. It’s important to note that the F here is NOT the same F as the one in F = ma. The F in F = ma is the \*resultant\* force, the sum of all the forces acting on an object, while the F in this equation is just the force due to the gravitational pull of one planet.

This means to get the resultant force acting on a planet, we must compute for that planet and all the other bodies in the system. We then add those forces up (some may be negative and cancel each other out) to get the resultant force. This is then given to F = ma, and the acceleration from that is given to Verlet integration.

To summarise everything covered in this section, a flow chart on the next page describes what happens to a single planet on each tick of the program:



### Collisions

First, why do we need to model collisions in a solar system? Planets in our system do not hit each other very often, if Mars had collided with us we wouldn’t be in a very good spot and no one would be reading my NEA analysis right now. However, the client requested that the mass of the planets be something you could change.

If the mass of the planets is changed, that will disturb the system, and could lead to collisions. It’s very likely that a child will set the mass of a planet to be very large for fun, and cause everything else to fall towards it. This would lead to objects overlapping, which leads to some very bad errors. Remember how force was GMm/r²? If r, the distance between objects, was 0, we would have a division by 0, and python would not like that.

The only site that my client mentioned that allowed you to change the mass did not handle collisions. Instead, it just stopped the simulation when the planets touched to avoid division by zero errors. Personally, I found this disappointing when playing with the simulation, and I expect other students would feel the same way, so I want to handle collisions realistically. Now, let’s cover how to handle them:

There are two important quantities that are always conserved in collisions, momentum and energy. Momentum, which is the mass of an object times the velocity of an object, is ALWAYS conserved. This means that if one object gains momentum, meaning it speeds up, the other one has to lose momentum to compensate, so it slows down. This gets more complicated when objects reverse direction, this will be covered in the design section as the maths can get quite technical.

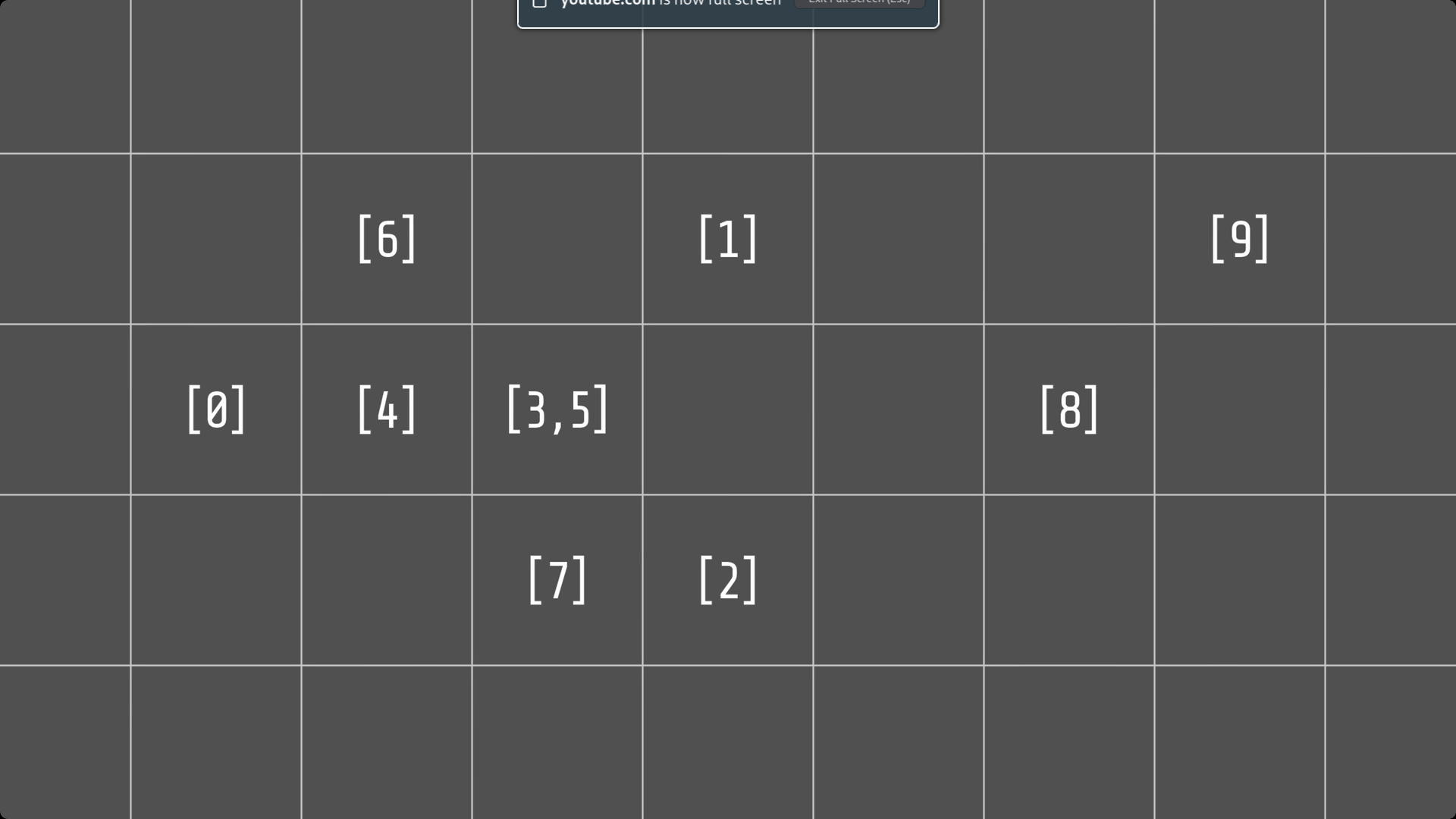
Energy, on the other hand, is a bit different. There are different types of energy. In collisions, typically what happens is that some of the kinetic energy (the energy objects have from being speedy) is lost and converted to other forms of energy such as sound and heat. The amount of energy lost to other forms is measured by a number, which we’ll call . When then no energy is lost, and when = 0 all energy is lost and objects that collide slow down until they lose all their speed. I will set as a parameter in the setup of the solar system, teachers and students will be able to change how bouncy their planets are. I will probably call it something like “bounciness of planets” rather than its proper name, coefficient of restitution, as it’s a further maths A level topic that most year 7s will not know about.

### *Optimisation*

The only concern with collisions is optimisation. Checking whether or not planets collide uses an algorithm. For anyone who hasn’t done big-O notation, what that means is that if the amount of planets doubles, the time taken for the simulation to do all the calculations quadruples. This is bad. However, there are many optimisation tricks available. The simplest one is to use “cells”, here’s a diagram:



Each one of those translucent gray circles is a hypothetical planet, and they’ve been numbered 0-9. What we now do is take their centres (yellow dots) and check whats square of the grid their centres are in. Each centre has a list associated with it. The lists for the diagram above will look like this:



Now, rather than checking whether one planet collides with every other planet in the simulation, we only check whether that planet collides with planets in its cell or adjacent cells. This is a massive performance improvement.

### Objectives

1. Be physically accurate.
   1. Model gravity accurately
   2. Model collisions accurately
   3. Conserve important quantities (momentum and energy)
2. Be hosted on a website
3. Adequately explain concepts requested by client
   1. Explain and show satellite orbits
   2. Explain days, seasons and years
   3. Explain how Newtonian physics connects to the solar system
      1. Include useful diagrams such as force diagrams to aid with this
4. Display information / links to reliable sources when planets are clicked on
5. Include a size comparison
6. The mass of planets should be editable
7. It should be possible to create your own solar system
8. Load someone else’s solar system from file
9. Access NASA JPL’s Horizons system to automatically import data on planets
   1. Use said data to load our real solar system into the program
10. Be sufficiently optimised and not suffer frame drops

### References

Cells diagrams in the optimisation section of collision come from this video here:

<https://www.youtube.com/watch?v=9IULfQH7E90>