Malvern College

Contents

[Analysis 2](#_Toc164427277)

[Introduction to the Problem 2](#_Toc164427278)

[Suitability of a Computational Solution 2](#_Toc164427279)

[Features of My Solution 3](#_Toc164427280)

[Analysis of current solutions 3](#_Toc164427281)

[Summary of Analysis of Current Solutions 13](#_Toc164427282)

[Stakeholders 14](#_Toc164427283)

[Interview Questions 15](#_Toc164427284)

[Interview Response 16](#_Toc164427285)

[Analysis of Interview Responses 19](#_Toc164427286)

[Decomposition 20](#_Toc164427287)

[Abstraction 20](#_Toc164427288)

[Essential Features of Solution 20](#_Toc164427289)

[Requirements 21](#_Toc164427290)

[Stakeholder Requirements 21](#_Toc164427291)

[Additional Requirements 22](#_Toc164427292)

[Success Criteria and Methods of Validation 22](#_Toc164427293)

[Limitations of the Simulation 25](#_Toc164427294)

[Hardware and Software Requirements 25](#_Toc164427295)

[Design 26](#_Toc164427296)

[Algorithms Overview 26](#_Toc164427297)

[Structure of the Solution 28](#_Toc164427298)

[List and Explanation of Algorithms 28](#_Toc164427299)

[Testing These Algorithms 36](#_Toc164427300)

[Graphical User Interface 37](#_Toc164427301)

[Explanation of User Interface Choices 40](#_Toc164427302)

[Usability Features 41](#_Toc164427303)

[Key Variables/Key Data Structures/Key Classes 42](#_Toc164427304)

[Explanation and Justification of This Process 44](#_Toc164427305)

[Development 45](#_Toc164427306)

[Stage 1:Implementation of the Vector Class 45](#_Toc164427307)

[Review of Stage 1 53](#_Toc164427308)

# Analysis

## Introduction to the Problem

I was inspired by the British Computational Physics Olympiad to undertake this problem. The problem involves simulating the solar system and the orbit of planets within it. I thought, why can’t we have any solar system with any gravitational constant? And decided to make this my project.

The target is to create an application/website that allows a user to create their own solar system. They should be able to add planets, change their masses, change the gravitational constant and see what happens. This is a tool that can be used in education, for example teaching pupils astrophysics in A level physics. I take A level physics and so this is a good practice for me to learn more about physics as well.

The system should use a GUI to allow for user to make/edit their solar system with a way of changing it as per described in the above paragraph, and should simulate and display the orbit of those planetary bodies.

The system could have a way of storing different solar systems created by the user, storing data about each planetary bodies’ current position, velocity and acceleration, as well as constants such as their mass. Ideally the system could make use of webserver tools to host itself on a website where people can use this. For adding complexity the system could also make use of login systems and allow other users to view each other’s solar systems and simulate their orbits.

The reason why I am making this is because I think that in education we can use more simulation based things to demonstrate the theory behind how things work. In physics, particularly astrophysics, it is often times quite hard for students to see or visualize what is going on, and so I want to develop a tool to aid students in learning.

## Suitability of a Computational Solution

This problem requires calculation of acceleration due to the resultant force of gravitational attraction due to the masses of each planet the user adds. This solution needs to run on a computer for the simple fact that no one is capable of doing it faster than a computer, as you need to do multiple integrations to calculate the position vector of a planet at time t, and t ideally needs to be a small number for a suitable degree of accuracy. The solution may calculate the acceleration every 0.1 seconds, as it is impossible to find the acceleration for a time step that is approximately 0 in real time. There are no real way of implementing this without a computer.  
  
Moreover, the user will interact with the solution to add more planets and use user interfaces to change settings, which is quite hard and unthinkable of to implement with anything other than a computational solution.

The problem Involves a many-stage calculation that require first finding the position vector of each planet, then using Newton’s law of gravitation to find the force planet A exerts on planet B due to gravitational attraction, then dividing by the mass to find acceleration. Once you have acceleration you need to integrate to find velocity and integrate again to find displacement. This makes it suitable for decomposition, which I will talk about how it applies to my solution later. It is also suitable for abstraction to take place since I will not be modelling any unnecessary parts of the solar system. Due to the two mentioned factors I wish to justify why this solution is suitable to a computational solution.

## Features of My Solution

To summarize the current problem, the program needs to have a GUI for user to add more planets, change the masses of planets and stars, set a time step to run at (e.g. 1 second in real life = 1 year in simulation), and see the orbit of the overall system.

Ideally the solution should have a means of storing what the user has created permanently but is not the main focus of the problem. The program could possibly have a login system and a way of viewing other people’s simulations.

The above are all features that I can think of, and I will consult with stakeholders for further input. To summarize, the features are listed below:

|  |
| --- |
| Have an intuitive GUI for the user to interact with my solution. |
| Have a setting or a way to change the masses of planets and stars |
| Have a setting or a way to change the time step it runs at (e.g. 1 second = 1 year) |
| The user can see the orbits and movement of planets simulated. |
| (Extension) – The user can save what they have created |
| (Extension) – The user can view what others have created |
| (Extension) – Webserver hosting to make it hosted on a webpage |
| (Extension) – Implement thick client and thin server |

More features are to be added after interviewing stakeholders.

## Analysis of current solutions

**List of current solutions (query result from google):**

1. UCL Planetary Orbit Simulation

<https://www.ucl.ac.uk/~zcapg66/work/COMP4/simulations/orbit/orbit.html>

2. UNL Astronomy Education Orbit Simulation

<https://astro.unl.edu/classaction/animations/renaissance/kepler.html>

3. JPL Solar System Dynamics

<https://ssd.jpl.nasa.gov/tools/orbit_viewer.html>

4. planetaryorbits.com

<http://www.planetaryorbits.com>

5. Sebastion Lague’s Coding Adventure (youtube)   
<www.youtube.com/watch?v=7axImc1sxa0>

Analysis of solution number 1 (UCL Planetary Orbit Simulation):

Introduction:

This is a software developed by UCL and this is an orbit simulation I found on their website.

Overall view:   
A screen shot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated

A screenshot of a game

Description automatically generated

List of features

|  |
| --- |
| A display to show the distance of mouse from sun |
| Left click to create anticlockwise planet |
| Right click to create clockwise planet. |
| Mass of the sun can be changed |
| Can display data of planets such as acceleration, speed, and radius. |
| Can pause/play and reset simulation |
| Can change the colour of the planets created. |

While the solution has many features that are suitable for my solution, and the GUI looks clean, there are some missing features that I want to demonstrate. The solution shown above uses a 2-body simulation method that utilizes Keppler’s laws of planetary motion which does not apply for a N-body simulation. My solution requires using N-body simulation methods and so I cannot use the same approach as theirs. They are also missing a way of changing the mass of planets, which they have justified as negligible in a 2-body simulation. The mass of each planets play a significant role on a N-body simulation with non-fixed orbits. However, I like how they can display data of planets, pausing and playing the simulation.

Summary of solution number 1 (Pros):

|  |
| --- |
| Has intuitive GUI |
| Can add planets which is what I want |
| Can change the mass of the sun |
| Can display data of planets |
| Can pause and play |

Summary of solution number 1 (Cons):

|  |
| --- |
| It is a 2-body simulation, which means that it does not apply to my solution |
| Cannot change mass of planets |
| Orbits are fixed |
| Cannot zoom in or zoom out |
| Planets can phase through each other |

I can take inspiration from their UI design as it is quite simple and I think I can make it quite easily. They have a well-documented webpage demonstrating how to use their application and so I think I could do something similar to that. However, I still need to come up with an algorithm myself as theirs does not apply to my solution. The zoom in and out feature is something that I can implement as I will be changing the orbits of planets and those can be quite big. I need to allow for the user to see the whole system or zoom into a specific part. Overall the simulation ran pretty fast with an average of 60 frames per second, and I want to achieve something similar for a smooth end user experience.

Analysis of solution number 2 (UNL Astronomy Education):

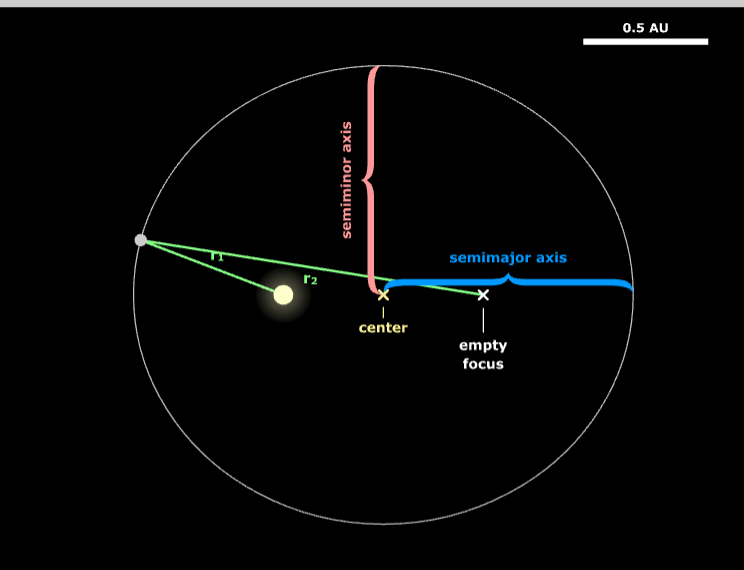
Introduction:

This is a website that hosts a simulation on it. I found this website on google when researching similar simulations such as the one by UCL above.

Overall view

A screenshot of a computer

Description automatically generated

A screenshot of a graph

Description automatically generated

A graph and diagram of a function

Description automatically generated

List of features

|  |
| --- |
| Ability to set parameters to a known planet |
| Changing semi-major axis |
| Changing eccentricity |
| Changing animation rate |
| Showing information about orbit such as empty locus and distance from loci |
| Plotting of areas swept in a certain time to demonstrate Keppler’s second law |
| Plotting of period against semi-major axis to demonstrate Keppler’s third law |
| Visualisation of the solar system’s orbits |

This solution has many features that allow for educational purposes. It successfully demonstrates many astrophysics laws such as Keppler’s law. It has a lot of UI going on which may be confusing for the casual user. If there is a way of simplifying the UI then it would be great. I think that overall this is a really good solution but unfortunately this solution does not apply to my scenario because, once again, it is only a 2-body simulation.

Summary of solution number 2 (Pros):

|  |
| --- |
| Lots of features to demonstrate physics |
| Intuitive UI if you know what you are doing |
| Many features to customise the simulation |
| Adaptive zoom that adjusts based on the size of the orbit so that it is always viewed in full, and in a reasonable scale. |
| Can pause and play |

Summary of solution number 2 (Cons):

|  |
| --- |
| GUI may be confusing if you don’t know what the terms mean, though you can easily find out by testing what happens if you move the sliders |
| Does not apply to my scenario |

Overall this solution is really good and once again I can use the GUI as an example as to how I might design my own. However, one problem with this is that it does not achieve what I want to do – to allow users to add more planets and simulate N-body scenarios. The amount of features and educational capabilities is something that I can implement in my solution.

Analysis of solution number 3 (JPL Solar System Dynamics):

Introduction:   
  
I found this website on google, and the 3d aspect of it is something that is worth looking into. It is a bit laggy when you drag and change the camera angle. Possibly because I have not turned on hardware acceleration in my browser.

A screen shot of a computer screen

Description automatically generated

List of features

|  |
| --- |
| 3D plot of orbits |
| Viewing of current position of planets (approximation) |
| Clicking on a planet allows you to centre your camera at it. |

I think that this solution is a little bit weaker compared to the other ones as it misses quite a bit of features that the other two both have. The UCL one has the ability for user to add more planets which I want, and the second one has a lot of features for you to plot and change things. It allowed the user to change the orbits and experience Keppler’s three laws. However, for this solution you can only view the orbits. Despite being in 3d, the lack of the features made this a comparatively worse solution to my problem.

Summary of solution 3 (Pros):

|  |
| --- |
| In 3 dimensions |
| Flexible camera, as you can change centre to focus on each of the planets |
| Intuitive, as if you drag you see the camera angle change. |

Summary of solution 3 (Cons):

|  |
| --- |
| Too little features for the user to interact with the system |
| The camera is a little bit jittery and made me feel dizzy |
| Although it is in 3D, it is really hard to tell from a glance, and depth perception is hard due to the weird design of the axis. |
| No way of adding more planets or altering the orbits |
| Doesn’t really allow any demonstration of physics (in other words we can’t do much with it) |
| Slow bootup time |

Overall this solution is less suitable for my solution. I can consider what they did badly with 3 dimensions and try to see what I can do if I choose to move on to 3 dimensions after creating a working example with 2 dimensions. I also want to avoid the long start up time that slowed down the user experience.

Analysis of solution number 4 (planetaryorbits.com):   
  
Introduction:   
  
Yet another website I found on google, this one has more scientific graphs and different view angles. Slightly faster than the 3d one in terms of startup time.

A screenshot of a video game

Description automatically generated

A line with a line in the middle

Description automatically generated with medium confidence

A screen shot of a graph

Description automatically generated

List of features

|  |
| --- |
| 2D view from above and sideways. |
| View of planets from earth with variation of time. |
| Zoom in and out (not really flexible, as you can only zoom in to the inner solar system and zoom out to view the entire solar system) |
| Helpful GUI to show the user what they need to do to zoom in and out, or to change the view. |

This solution suffers the same drawbacks as solution 2. It only shows the solar system and does not allow the user to dynamically change the masses of planets and stars, and it does not allow the user to add more planets. The orbits are fixed and cannot be changed, though it allows for a way for the user to view it from different angles. When compared to the other solutions, this one is arguably the weakest as it has practically no features I am looking for and does not draw it in 3d, which ranks it lower than solution number 3.

Summary of solution number 4 (Pros):

|  |
| --- |
| Can view from different angles (though camera angle is static) |
| Has graph of stars and planets viewed from earth |
| Can zoom in and out |
| UI is pretty simple to use |

Summary of solution number 4 (Cons):

|  |
| --- |
| Uses a 2-body simulation to work out the orbits, presumably based on Keppler’s laws |
| Camera angle is static and fixed, and can only be changed by pressing a button. |
| Does not allow the user to zoom in and out using mouse wheel |
| No feature for user interaction with the orbits (the user cannot change the orbits) |

Overall I think what I can adapt from this solution is limited as most other solutions have enough features that I can try to incorporate in my solution. One thing I can learn not to do is how they manage camera. I can write it such that the camera can be moved with the mouse and the user can zoom in and out with the mouse wheel.

Analysis of solution number 5

Introduction:   
  
This is a youtube video that has been recommended to me by a teacher when I talked to them about my computer science project. It uses a N-body simulation using Newton’s law of gravitation and F=ma to calculate the position of planets using time steps.

A screenshot of a computer

Description automatically generated

List of features

|  |
| --- |
| User is able to freely add more planets |
| Able to change initial velocity of planets |
| Orbit is drawn upon creating planets, allowing for user to visualise what will happen without simulating it |
| The user can actually fly in a spacecraft to visit all of the planets |
| Planets can crash into each other and destroy one another. |

This solution is really cool and has lots of features that I am looking for. My end goal is basically something similar to what he achieved in this video. The user can create more planets and the system simulates the orbit of those planets. He managed to simulate a 3-body system in equilibrium. His approach was suited for a N-body simulation which is exactly what I want.

Summary of solution number 5 (Pros):

|  |
| --- |
| Uses a N-body approach for simulation |
| Seems to run pretty fast |
| Has a way of seeing the orbit before simulating, allowing for the user to make changes without having to run the simulation |
| Graphics are nice, but not sure if they have a way for a user to add more planets while playing the game. |

Summary of solution number 5 (Cons):

|  |
| --- |
| Doesn’t seem to be focused on letting the users create planets, rather is focused on having the user explore a fixed solar system. |
| Graphics may be unnecessary in my situation, as I am only providing a simulation, not a video game. |
| Initially he had problems with the physics breaking as he moved to the rim of the solar system, but he adjusted for that by moving the whole solar system around the user’s loaded area to avoid issues with memory. |
| Not sure if he used a time-step method, but if he did, then I should improve on it because it requires a time step that is extremely small for accuracy and it allows for the system to drift substantially over time. |

What I can take on board from his solution is the N-body approach. He used Newton’s law of gravitation for each planet to calculate acceleration then used a small time step to use during integration to estimate the positions of each planet. I can use the same algorithm to produce a similar result in 2D for my solution.

## Summary of Analysis of Current Solutions

**Identification**

So far, we see a few different approaches. One main approach I see in two of the solutions is to use Keppler’s laws of motion for 2 body simulations. They have used them to determine the position at any time given the semi-major axis, semi-minor axis, and eccentricity.

**Justification**

This is not a suitable solution to my problem, and I wish to justify this claim by saying that the goal of my project is to simulate a N-body situation. This feature of my problem makes this approach unfeasible.

**Identification**

Another approach was just to plot the orbits using currently known data and not allow the user to interact with it.

**Justification**

Again, this approach is unsuitable for my solution as I wish to create a system/application where the user can dynamically modify the system as needed (e.g. changing mass of planets, adding more planets). While the first two solutions which use the first approach have these features, they do not allow for a N-body simulation. The latter two solutions use this approach, but completely fail to meet the needs as specified earlier.

**Identification**

A third approach I have seen is from solution number 5. They have used a N-body approach for calculating acceleration then used a time slice approach for estimating the positions of planets. While this approach is suitable for my solution, I can make it better by solving a higher order of ordinary differential equations using algorithms such as leapfrog integration to ensure that there is no deviation over large amounts of time, and that I do not need to use an unnecessarily small time step which increases computational cost.

**Justification**

This approach is suitable for my solution as it is basically what I want. They have used a N-body approach which is suitable for allowing the user to add more planets and simulate the effects of gravity due to each of the planets.

## Stakeholders

There are a few stakeholders that may be interested in my solution. For example, a physics teacher may be interested in the solution as it can be used in education. They may use the solution to demonstrate the laws of astrophysics. Another stakeholder may be students who are interested in physics and want to use a visual way of learning about astrophysics, or students who are interested in computer science/maths and want to learn more about the backbones of my solution.

Analysing and summarizing what a physics teacher may use the solution for:

|  |
| --- |
| Teaching astrophysics |
| Demonstrating effects of gravity |
| Providing a visual way of learning for students |
| Research into planetary motion |

Reasons why my solution may be appropriate for a physics teacher’s needs:

|  |
| --- |
| An interactive simulation is likely to be helpful when teaching pupils about astrophysics. The teacher is able to change the orbit of planets and this allows the teacher to ask students for key terms such as the semi-major axis of the orbit. The teacher may ask students to draw in axis to demonstrate their knowledge. |
| My simulation aims to be able to change the gravitational constant and simulate its consequence, as well as changing the mass of planets. This is appropriate for the need of demonstrating the effects of gravity as the force of gravity exerted on each planet depends on the mass of the planet that is exerting the force as well as the gravitational constant. Hence my solution may be appropriate for this need. |
| A simulation is in itself a pictural of showing information. This allows for a visual way of demonstrating physics theories such as Newton’s law of gravitation. This means that the solution is highly likely to meet the need of providing a visual way of learning for students. |

Analysing and summarizing what a student may use the solution for:

|  |
| --- |
| Learning astrophysics |
| Entertainment – creating a solar system may be considered fun for some |
| Learning visually as mentioned before |
| Research into planetary motion and the mathematics behind it |

Reasons why my solution may be appropriate for a student’s needs:

|  |
| --- |
| It provides a graphical way of learning about astrophysics |
| Visualisation of solar systems can be entertaining and visually pleasing. |
| It is a good way to learn visually. |
| There is quite a lot of mathematics behind it which I can make the program demonstrate simply, to enhance the learning. |

### Interview Questions

For learning more about my demographic I will be using the following questions.

1. If you had an application that lets you simulate the solar system, add planets, remove planets, and adjust attributes of the planets such as mass and velocity, what would you most likely use it for?
2. Would you prefer a simple to use application with little features, or an application with lots of different features but slightly harder to understand and use?
3. Would it matter if you can store the solar systems you make and load it back?
4. Would it be great if you can load what other people have made?
5. How accurate would you like the simulation to be? Would you rather have it run faster at the cost of accuracy, or be more accurate at the cost of being slower?

Question 1 asks the stakeholders what the use of the application might be. For example they may say that they would use it for teaching astrophysics, and then I can try to tailor it for an educational purpose with explanations to show the physics and mathematics behind it.

Question 2 asks the stakeholders for their preference of ease of use. It helps me understand how complex the simulation should be. It may not be useful to simulate complicated physical scenarios such as nebulas by each individual molecule if the user does not want to deal with setting up a nebula with the application, and so they may be omitted. Some features that are not core may be tucked away into a separate menu to make it more accessible.

Question 3 confronts the stakeholders about the necessity of a permanent storage. I suspect that most of them would agree that it is quite important to store your work, as it may take some time to get the simulation right and for planets not to crash into each other by changing the initial velocity. It may be a useful feature. But if most of my demographic thinks otherwise then I may omit the permanent storage to focus on improving the simulation.

Question 4 asks about the need for a way to view other people’s work. I think that this is a good feature to have, as teachers may then be able to send stuff to students for them to have a play with it, to learn on their own. However, once again, if the stakeholders think otherwise then I may omit it to focus on other areas of the project.

Question 5 deals with the accuracy vs speed problem of a simulation. I basically want to know how accurate I need to make my simulation, and how fast I need my application to run at. If I use a different algorithm I may get a faster application at the cost of accuracy of the simulation. One example is if I use leapfrog integration or if I use a different order of ordinary differential equations to calculate the position vector of a planet at any time. Depending on which one I use I get more accurate results but use more time. The stakeholders may prefer to have a faster application that has a good level of accuracy that can roughly show the physics.

All of the above questions fulfil the basic knowledge I need to have about the demographic and I can analyse from the results what the stakeholders want from my solution and thus what I need to do in order to achieve them.

### Interview Response

Lisa, a student interested in physics.

[Me]: Hi do you have 5 minutes for a few questions. This is for my coursework for compsci.

[Lisa]: Yeah sure I have 5 minutes.

[Me]: If you have an application that lets you simulate the solar system, what would you most likely use it for?

[Lisa]: Science projects or competitions when I need to use a graph.

[Me]: Okay. Would you prefer to have an app that has lots of features but feel harder to use or an app that has simpler features but is easier to use?

[Lisa]: While I like having different features I think that, um, simple features may be better if it makes more sense to me.

[Me]: Would it matter if you can store solar systems and then load it back?

[Lisa]: Yes.

[Me]: Would it be great if you can load what other people have made?

[Lisa]: Yes.

[Me]: How accurate would you like the simulation to be? Like totally accurate but kinda slow, or not so accurate but fast?

[Lisa]: More accurate but at the cost of being slow.

- End of interview –

Steven, a teacher of physics

[Me]: So for my coursework I am planning to do something similar to this [shows solution number 2], where you’ve got like a star, planets orbiting around it, and then I want to be able to change the mass of the planets, change the mass of the sun, and then have this be moving as well.

[Steven]: Ok. By having this be moving you mean have the star moving as well?

[Me]: Yeah yeah, so all of the planets and stars affect each other, and then they move around. And then but this one uses Keppler’s laws of motion which does not apply for a N-body simulation. So basically the questions I have is if you have an application that lets you drag and drop and adjust the mass of the planets to see the orbits, what would you most likely to use the application for.

[Steven]: Uh, interesting. What would I most likely use it for…There are…there’s some GCSE topics that are always nice. There is a simple simulation by Phet. So maybe if you go on to Phet. [I tried to look it up] Well you don’t have time to look now you can just talk about that and what causes the change in the gravitational forces. We can do something like “let’s change the mass of the sun and see how it affects the orbits around the planets” and you can do the same for the Earth and the moon surrounding it. It can have a trail of where it’s recently been to trail the orbit so you can certainly use it to introduce gravitational fields to GCSE classes. And then also for Newton’s law of gravitation for your sixth form classes. And I’m just thinking now, I wonder if you had some values on there as well, like if it actually showed the gravitational force of attraction between two bodies. Then you could get them [students] to actually vary the masses and the distance between them and to maybe see if they can figure out the Newton’s law of gravitation and maybe what the factors are—

[Me]: So you want like, a setting where you can press [select] two planets and then show the force of A exerted on B and B exert on A?

[Steven] Yeah, yeah that would be quite useful, if you can change the distance or mass.

[Me]: I think I can let you like change the initial position of the planets by dragging it around and then you can start the simulation and see the values.

[Steven]: If they could change the distance, they [students] could at least figure out that the force is inversely proportional to r [radius of orbit] squared, like that would be a value right.

[Me]: And then my second question is: would you prefer a simple application with less features or lots of different features but might not like, seem very intuitive to use?

[Steven]: Well, what sort of extra features do you have in mind?

[Me]: Like if I look at the second solution I’ve got, you’ve got so many like different menus and settings. You can change the semi-major axis, you can change its eccentricity and stuff. But you can set it to show these [radial lines from loci, refer to solution number 2]. So when it moves, the lines actually follows it [the planet]. And you can have a setting to show the area. So this one is more based on Keppler’s laws but then I think there is like too many things going on? And it can be confusing if you are trying to learn one thing at a time or something.

[Steven]: Yeah, to be honest, I wonder…is it okay to say both? ‘Cause you can have…I keep going back to these Phet simulations again…you can have a basic layer or you can even have just, for example for circuits, they have “basic” and then “lab”. So yes, if I just want to teach kids about if I increase the mass then what’s going to happen there. Or you know, if we’re doing sixth form then we do want to have an advanced mode.

[Me]: Maybe I can tuck away all of the advanced stuff into like an advanced menu where you can change more stuff but then have like a more simple GUI for the sake of being simple at first.

[Steven]: Yeah absolutely, that’s alright.

[Me]: Ok. And then my third question is, let me just scroll down: do you think it’ll be better if you can like save what you’ve created and then load it back, or is that like not really necessary?

[Steven]: It depends on how easy it is to do...if I am completely honest, I think we’re unlikely to go back to exactly what we did before. Unless, you know that functionality of can we figure out the how of things, like can they figure out r^2, there may be a need of collecting a lot of data and go back to that. If it’s easy sure, but otherwise they can just do it in the lesson and store their data elsewhere. So yeah I wouldn’t say that’s a huge priority.

[Me]: Ok. And because that isn’t a huge priority, I assume that it won’t be a, very important for you to be able look at what other people have made?

[Steven]: No, I don’t think that needs to be on the software. I think-

[Me]: You can just go and look on the screens of the students.

[Steven]: Exactly so even we as a class, I use One Note all the time. But if I were to get you guys [my class] to do your work on One Note, and then try to mark it on that software, it just makes everyone’s life harder. And it’s also harder to show other people the work, which obviously isn’t the most important thing, but it is important.

[Me]: Ok so one final question is like: between the speed and the accuracy of the application, which one do you prefer? Do you prefer the application to be like very accurate at the cost of it running a little bit slow, or do you want it to run fast at the cost of being less accurate?

[Steven]: I think it depends on how extreme those values are, right? I think as people we are getting used to things loading almost immediately these days, and there is a cut off time for people to say this is too slow. And so long that it’s less than that. For these simulations, you [most people] are happy to wait for it to load for, 10 seconds? And so anything within that- I bet you there’s a value out there that people have researched and said right that is the cut off time.

[Me]: That is the point where people get annoyed.

[Steven]: Yeah where the general public get annoyed. But for these Phet simulations, some of them do take a little while to load. If you can get the values [experimental values] you are looking for as a result then I think it’s okay. I think 30 seconds is way too long, and 10 seconds is absolutely fine.

[Me]: Ok, that’s it. Thank you very much.

- end of interview –

### Analysis of Interview Responses

Looking at the response from Lisa, I think that the end users of this application may be used for science-related projects or competitions. She also thinks that it’s better to have simple features to not make everything seem cluttered. She would like to be able to save and load what she has done. One interesting point that contradicts my initial prediction is that she would rather the simulation take more time and be more accurate.

Steven thinks that the application can be used in an educational setting, which I agree. A new point that was made was that I can perhaps make a setting for the user to see the effects of gravity of one planet on another, when the user selects a pair of planetary bodies. While Lisa thinks that she would prefer having less features, Steven thinks that I can have the best of both worlds by hiding away the advanced settings if it may be too confusing. He also believes that loading/saving creations and seeing other people’s solar systems may not be entirely necessary in an educational environment, as teachers can just look at student’s screens physically, and it may in fact make it more inconvenient if they had to do it over the software. Finally he thinks that it is okay to have the simulation load for approximately 10 seconds but any more than that is not acceptable.

Key takeaways from the stakeholder interviews are listed as follows:

|  |
| --- |
| Purpose of the application is for science and education, so I should try to make it useful for that |
| Simple features at first seem appropriate, and then more capabilities of the application may be accessed in a different mode or hidden in a menu that users can click on if they so wish to, such that pupils from an earlier key stage may still be benefitted by the application |
| Saving and loading solar systems created may be useful for students exploring on their own but may not be as useful for teachers. |
| Stakeholders seem to prefer accuracy over speed of the application. Optimisation will be required if I use an accurate simulation, to ensure that the speed is satisfactory while providing a high level of accuracy. |
| An additional point that is not intended from the interview is that it may be good to have settings for the user to select two planets and see the interaction in terms of the force they apply on each other. |

## Decomposition

The problem is suitable for decomposition as you need to apply step-by-step algorithms to achieve the target. An initial view of what the steps I may take are as follows:

* Taking user input to add new planets and place it on the mouse cursor position.
* Allowing the user to drag and drop the planets to where they want it.
* Finding the force other planets exert on the planet.
* Dividing by the mass to find instantaneous acceleration.
* Using either a time-step approach (using very small time steps and calculating acceleration at those time steps then integrating to find velocity and position) or use a differential equation numerical method to solve for new position (such as the leapfrog integration method)
* Changing the attributes of planets (position, velocity)
* Updating the display of the screen to show the planets.
* Detecting if the user pushes the button to pause the simulation.

The entire project together may seem daunting but in their own are doable tasks. This justifies how it is suitable for computational method, such as decomposition.

## Abstraction

The problem involves physical calculations and that might cause some computational problems if I try to simulate everything that is going on in the universe to the molecular level. I can apply abstraction in this case and say that I will ignore what goes on down to a molecular level and focus on how it works on a larger scale that applies to my situation. I have decided to model the planets as perfect spheres which is not the case in reality, but for a simulation this is good enough. I will also model the planets as perfectly uniform with the mass at the centre of the spheres, which is not the case in reality which would affect the accuracy of the model. But since I am after a simple application that demonstrates the effects of gravity and mass, I can ignore the effects of the masses not being uniform, and whatever that happens in the molecular and quantum level.

## Essential Features of Solution

The following are the essential functionality of the solution:

|  |  |  |
| --- | --- | --- |
| Name | Justification | Priority |
| The program shall be able to plot the orbit of planets | The whole purpose of the project is to do this | High |
| The program shall be able to change the mass of planets | The project’s purpose is to allow for interaction with the orbits | High |
| The program shall be able to change the gravitational constant | The aim of the project is to have a mean of interacting with the gravitational constant | High |

Some of the essential features listed here can be further split into different criteria which I will define in my success criteria section.

## Requirements

### Stakeholder Requirements

From what I gather from the interview, the requirement from the stakeholder’s can be summarized as follows.

|  |  |  |
| --- | --- | --- |
| Name/Description | Justification | Priority |
| The program shall have simple features at first glance, and this includes being able to adjust the distance from each planets. | As suggested by stakeholders. Not as important as the above objectives which are the crux of the project, but still quite important | Medium |
| The program shall have an advanced setting for more display or adjustments such as selecting two planets and seeing the force they exert on each other | As suggested by Steven | Medium |
| The program shall have a way of displaying the force a pair of planets exert on each other | As suggested by Steven | Medium |
| The program shall prioritise accuracy over speed | As extrapolated from the summary of interviews. This objective is not as important as if the overall program works then it is fine, but this enhances the user experience. | Medium-Low. |
| The program shall be accurate to around 10% of the actual value | Accuracy is important as per above objective. | Medium |

That is all of the objectives I can extrapolate from the stakeholder interviews.

### Additional Requirements

Listed here are some additional/optional requirements that the project may need to meet.

|  |  |  |
| --- | --- | --- |
| Name/Description | Justification | Priority |
| The user interface shall be intuitive to use and not confusing | Enhancement of user experience. Not a main priority | Low |
| The program shall have a means of storing what the user have created using the software | As explained by Steven, this is not an important part of the project, and the user may not use this. This can be an extension to the project for anyone who may use it, such as Lisa. | Low |
| The program shall have a means of sharing what the user has created | As explained by Steven this is also not an important feature of the program. However, as users like Lisa may want to have this feature, this could be an optional feature for me to implement after the overall project hits the necessary requirements first. | Low |

### Success Criteria and Methods of Validation

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | Justification | Validation | Optional? |
| The program shall be able to plot the orbit of planets | Overall goal of the project | Screenshot of plot | No |
| The program shall be able to calculate the orbits of planet | Required in order to plot | Screenshot of different plots with different values for masses | No |
| The program shall be able to subtract two vectors. | Needed for calculating the orbits. | Test with test data. | No |
| The program shall be able to find the distance between two vectors. | Needed for calculating the orbits. | Test with test data. | No |
| The program shall be able to divide a vector by a scalar | Needed for calculating the orbits | Test with test data | No |
| The program shall be able to find the unit vector between two vectors | Needed for calculating the orbits | Test with test data | No |
| The program shall be able to find the force a planet exerts on another | Required for calculating the orbits | Test with test data | No |
| The program shall be able to find the resultant force on a planet | Required for calculating the orbits | Test with test data | No |
| The program shall be able to find the acceleration of a planet | Required for calculating the orbits | Test with test data | No |
| The program shall be able to find the position vector of a planet a time step away | Required for calculating the orbit | Test with test data | No |
| The program shall allow the user to change the orbits | Overall aim of the project | Screenshot of user interface.  Video of user using program to change orbit. | No |
| The program shall allow the user to change the mass of planets | Overall aim of the project | Screenshot of user interface  Video of user interacting with the menu to change the mass | No |
| The program shall allow the user to change the gravitational constant | Overall aim of the project | Screenshot of user interface Video of user interacting with menu to change the gravitational constant | No |
| The program shall be accurate within 10% of actual value | Accuracy is more important than speed, as justified in the stakeholder analysis | Plotting of known planets and cross-comparison.  Using the program to calculate the error using known formulas for the algorithm that I use. | No |
| The advanced settings such as gravitational constant shall be tucked away into an advanced menu that the user can open | To prevent cluttering of the main window, and optimising the user experience as justified in the stakeholder analysis.  An optimisation objective, not a main objective. | Screenshot of user interface  Video of user interacting with the advanced menu | Yes |
| The program shall allow for the user to store what they have created | An option for the user to save what they created can be useful, but not necessary for the overall project as explained by Steven | A screenshot of the user interface A video showing the user interacting with the saving feature | Yes |
| The program shall allow for the user to share what they have created | The application may choose to use webserver hosting and user login systems to allow for the users to look at what other people have created | Screenshot of the user interface  Video of a user loading up what other people have made | Yes |
| The user interface shall be intuitive and easy to use | A clean user interface can make the user experience better.  An optimisation objective, not a necessity | Questionnaire/interview with stakeholders for opinion  Screenshot of the user interface | Yes |
| If implementing a user login system:   The database shall be secure and store the passwords as hashes | For security,  not very important for this task as it depends on whether I implement a system for multiple users | Screenshot of code  Screenshot of example password stored in database. | Yes |
| The program shall take no longer than 15 seconds to start up. | As discussed with Steven this may annoy the users. | Video of starting the application showing that it does not take longer to start | Yes |
| The program shall be able to allow for the user to log in to the system | Required for the users to load what they have created | Screenshot of login menu | Yes |
| There should be a play button for the user to press to start the simulation | Part of the GUI design | Screenshot of button | No |
| There should be a textbox or scrollbar for the user to adjust the mass of planets | Part of the GUI design | Screenshot of textbox/scrollbar | No |
| There should be a clear and concise tutorial about how to use the application | To illustrate how to use the software I developed | Screenshot of manual  Video showing how to use it | No |
| There should be a textbox or scroll bar for the user to change the gravitational constant | Part of the GUI design | Screenshot of textbox/scrollbar | No |
| The planets should be displayed as circles on the screen | Part of the graphics design | Screenshot of planets | No |
| There should be a button to open up an advanced menu for the user to adjust additional settings (if applicable) | Part of the GUI design, and something that I discussed with Steven | Screenshot of the menu | Yes |

### Limitations of the Simulation

The simulation is not all powerful. I cannot simulate down to the molecular level, and I cannot model the entire universe. Listed here are the limitations of the simulation.

|  |  |
| --- | --- |
| Limitation | Justification |
| The simulation will not be 100% accurate | It is theoretically impossible to model the position of planets continuously (e.g. with a time step that is basically 0), due to computational power |
| The simulation will not simulate each and every molecule separately | There is not enough computational power and I do not have enough knowledge to do so even when given the computational power |
| The planets will be modelled as perfectly spherical | To abstract away unnecessarily complex difficulties that comes with it.   It is not necessary for me to consider the unevenness of the planets, as the effect is negligible. |
| The planets will be modelled with their masses uniformly distributed within it. | To abstract away unnecessary complexities.  The effect of the distribution of mass is negligible in the scale/grand scheme of my simulation |
| The simulation cannot simulate infinitely many planets | There is not enough computational power to do so, as the computational cost increases exponentially with number of planets. If I had 3 planets and I wanted to add 1 to the system, I need to calculate 12 times, instead of 6 times. If I then add 1 more I need to calculate 20 times instead of 12 times. The increments increase further the bigger the number. This means that it becomes more and more difficult and slower as you add more planets in. |
| The simulation will not simulate the effects due to the spin of the object | To abstract away unnecessary complexities.  This project is aimed for a simple simulation, not a full on and in-depth simulation for 100% accuracy. This is meant for a quick demonstration of physics. |

### Hardware and Software Requirements

|  |  |
| --- | --- |
| Requirement | Justification |
| Windows 8 or later or equivalent operating systems, such as Mac or Linux | Requires python |
| 4GB of RAM minimum | Requires python, and requires pygame.  Pygame requires 4GB. |
| 1.5 GHz processor or equivalent | Need to be able to run python |
| 500 MB of storage | To store code |
| Has python installed | Project runs on python |
| Has pygame library installed | Project uses pygame to plot orbits |

# Design

## Algorithms Overview

The problem involves a couple of steps. I will break them down into different sets of problems for me to solve.

One of the set of problems I need to solve is the actual simulation itself. A step-by-step guide as to how I might tackle this problem is listed below.

|  |
| --- |
| Give a defined list of planets with their attributes defined (such as mass, position, velocity and acceleration). Define them as objects of the class planet. |
| For each planet that is in the simulation, I need to find the force each of the other planets exert on it.  To do this I can use the vector form of Newton’s law of gravitation.  Newton's Law of Universal Gravitation - ppt video online downloadsource <https://slideplayer.com/slide/273197/> |
| To find the force, I need to have the masses of the two planets involved, which can be accessed by the attribute of the planet object. (e.g. planet.mass). |
| I also need to find the distance between the two planets. I can do this by using:  Typed inside of desmos: <https://www.desmos.com/calculator>  Where I can get x1 by taking the first planet and then taking its .x value, and similarly for the rest of the variables. |
| Now that we have the distance between the planets, we need to use vector subtraction to find the vector r.  We can achieve this by doing (x1-x0)i + (y1-y0)j. This would give us the vector r using the standard unit vectors i and j. |
| Now we can find the unit vector r12, which is the unit vector that goes from object 1 to object 2 as shown in the diagram.  This can be done by dividing the vector r by the distance between them. |
| Now that we have the masses, the distance between them, and the unit vector, we can use the formula to find the force planet A exerts on planet B, given a value for gravitational constant. |
| Once I have the force of every other planet on one planet I am considering, I can add them all up to find the resultant force. |
| Once I have the resultant force, I can divide it by the mass of the planet I am considering to find its instantaneous acceleration. |
| Repeat for all planets in the simulation (it is this step that may be computationally expensive) |
| To find the next position I can apply leapfrog integration:  <https://en.wikipedia.org/wiki/Leapfrog_integration>  My acceleration is a function of all the position of the planets. I can write a method to find the acceleration of any planet at any time given all of the position of the planets, using the above steps.  I can then find the new velocity at half a time step away as it is the velocity at half a step before + the acceleration at the current step multiplied by the difference in time between time steps.  The new position is the current position + the velocity at half a step away multiplied by the difference in time between the time steps.  This is summarized in:  Source: Wikipedia, link is shown above |
| Now I just need to feed it to another algorithm for moving the planet to the according position.  I can achieve this by using libraries such as pygame to display the planets. |

Another set of problem is the setting up of the simulation. A step-by-step algorithm is listed below:

|  |
| --- |
| Have a button detect user input. If the user clicks on it, add a planet at the origin. |
| Initialise the planet to have attributes (mass, position, velocity and acceleration) to default values. |
| Have a button on the planet that detects click. If the user clicks on it open a menu that has settings for the user to change the values. |
| Use either a scrollbar or input box for values depending on complexity. If it is simpler then just use a input box. |
| Sanitise the input by ensuring that it does not exceed the limitations of the model (mass not too big, velocity not too big, etc) and does not contain code. |
| Once the user hits enter, check that the input does not contain any unallowed characters such as !@#$%^ and update the attribute of that planet accordingly. |
| Wait until the user hits play before beginning the simulation. |

Justification of why I chose to divide it like this:

|  |
| --- |
| Dividing it up makes it easier to solve the problem. |
| The separate steps combine together to form a complete solution to my problem. |
| Making the overall system more modular allows for easier testing, which leads to a more robust system. |

## Structure of the Solution

The project will be developed in a few different parts listed below:

* The vector class, most important because it is used by the planet class.
* The planet class, used for finding the position of the planets after a certain time.
* The GUI.
* Algorithm to handle the GUI.
* The main loop
  + Getting the user to enter the settings and planet details,
  + Putting the planets where they need to be,
  + Finding the resultant force on each planet,
  + Finding the new position of the planet,
  + Moving the planets to where they need to be,
  + Handling pauses,
  + Handing changing settings (need to pause simulation first)

## List and Explanation of Algorithms

A list of the algorithms I will use is listed below:

|  |
| --- |
| Vector subtraction |
| Finding distance between two points in a 2-dimensional grid |
| Dividing a vector by a scalar |
| Finding unit vector between two points in a 2d grid |
| Finding the force a planet exerts on another one |
| Finding resultant force exerted on one planet |
| Finding acceleration on one planet |
| Finding the velocity at half a time step away |
| Finding the position one step away |

**Vector subtraction:**

|  |
| --- |
| Input the x and y coordinates for the two vectors. In this algorithm we are taking away the first vector from the second vector. We are doing second - first , which would give us the vector that goes from the first vector to the second vector. |
| Subtract the x value of the first vector from the x value of the second vector. |
| Subtract the y value of the first vector from the y value of the second vector. |
| Define/instantiate a new vector with x value of the computed x value, and y value of the computed y value. |
| Return the new vector. |

Link to success criteria:

* The program shall be able to subtract two vectors.

Pseudo code for vector subtraction:

|  |
| --- |
| vector1\_x = integer( input() )  vector1\_y = integer( input() )  vector2\_x = integer( input() )  vector2\_y = integer( input() )   # enter all of the coordinates  vector3\_x = vector2\_x – vector1\_x  vector3\_y = vector2\_y – vector1\_y  return (vector3\_x, vector3\_y) |

**Finding the distance between two points in a 2d grid:**

|  |
| --- |
| Input x and y coordinates for the two vectors. |
| Use the vector subtraction algorithm as shown above, and store the returned vector. |
| Take the square of the x value and the y value of the vector. |
| Take the square root of the sum of the squares (of x and y) |
| Return the value |

Link to success criteria:

* The program shall be able to find the distance between two vectors.

Pseudo code for finding the distance between two points:

|  |
| --- |
| x1 = integer(input())  y1 = integer(input())  x2 = integer(input())  x2 = integer(input())  new\_vector = vector\_subtraction( x1,y1,x2,y2 )   # store the returned vector from the result of the vector subtraction algorithm  x\_square = new\_vector.x \*\* 2 # this is squaring it  y\_square = new\_vector.y \*\* 2  sum\_square = x\_square + y\_square # this is adding up the squares  return sum\_square \*\* 0.5 # this returns the square root of the sum of the squares  # return the square root of the sum of the differences |

**Dividing a vector by a scalar**

|  |
| --- |
| Input the x and y coordinates for the vector |
| Input the scalar you want to divide it by |
| Divide the x value by the scalar |
| Divide the y value by the scalar |
| Create/instantiate a new vector with the x and y values that have been divided |

Link to success criteria:

* The program shall be able to divide a vector by a scalar.

Pseudo code for dividing a vector by a scalar:

|  |
| --- |
| vector\_x = integer(input())  vector\_y = integer(input())  number = integer(input())  new\_x = vector\_x / number  new\_y = vector\_y / number  return (new\_x, new\_y) |

**Finding unit vector between two points in a 2d grid:**

|  |
| --- |
| Enter x and y coordinates for the 2 vectors |
| Take the distance between the two vectors using the algorithm above |
| Use vector subtraction to find a new vector that takes you from the first vector to the second vector. |
| Use vector division (for dividing by scalar) to divide the new vector by the distance between them |
| Return the vector after dividing |

Link to success criteria:

* The program shall be able to find the unit vector between two vectors.

Pseudo code to find the unit vector between two points in 2d.

|  |
| --- |
| vector1\_x = integer(input())  vector1\_y = integer(input())  vector2\_x = integer(input())  vector2\_y = integer(input())  distance = distance\_between(vector1\_x, vector1\_y, vector2\_x, vector2\_y)  new\_vector = vector\_subtraction(vector1\_x, vector1\_y, vector2\_x, vector2\_y)  unit\_vector = vector\_division(new\_vector, distance)  return unit\_vector |

**Finding the force one planet exerts on another one:**

|  |
| --- |
| Input the mass of the two planets |
| Input the positions of the two planets |
| Input the gravitational constant |
| Use the positions of the two planets and the algorithm defined above to find the distance between them |
| Use the unit vector algorithm to find the unit vector to go from one planet to the other |
| Use newton’s law of gravitation to find the force exerted by one planet on another planet. |
| Return the force as a vector |

Link to success criteria:

* The program shall be able to find the force one planet exerts on another.

Pseudo code for finding the force one planet exerts on another one:

|  |
| --- |
| mass1 = integer(input())  mass2 = integer(input())  planet1\_x = integer(input())  planet1\_y = integer(input())  planet2\_x = integer(input())  planet2\_y = integer(input())  gravitational\_constant = integer(input())  distance = distance\_between(planet1\_x, planet1\_y, planet2\_x, planet2\_y)  unit\_vector = find\_unit\_vector(planet1\_x, planet1\_y, planet2\_x, planet2\_y)  # goes from planet 1 to 2  force = -gravitational\_constant \* (mass1 \* mass2)/(distance\*\*2) \* unit\_vector  # this is using the formula given in the algorithms overview, in the picture  # this finds the force that planet 1 exerts on planet 2 as a vector |

**Finding the resultant force on one planet:**

|  |
| --- |
| For all of the other planets you need to: |
| Find the position of the planets |
| Find the distance between the two planets |
| Find the unit vector that takes you from the other planet to the planet you are considering |
| Use the algorithm above to find the force the planet exerts on it |
| Then you add all of the forces up |
| Return the resultant force |

Link to success criteria:

* The program shall be able to find the resultant force on one planet

Pseudo code for finding resultant force on one planet:

|  |
| --- |
| planet\_list = [planet1,planet2,planet3…]  # all of the planets should be defined as objects with x and y attributes. For now we will ignore the  # velocity and the acceleration as that is irrelevant to the force  # if we want to find the resultant force on planet1:  gravitational\_constant = integer(input())  resultant\_force = 0  for planet in planet list:  if planet is not planet1:  distance = distance\_between(planet1.x, planet1.y, planet.x, planet.y)  unit\_vector = find\_unit\_vector(planet1.x, planet1.y, planet.x, planet.y)  force = find\_force(gravitational\_constant, distance, unit\_vector)  resultant\_force = resultant\_force + force  return resultant\_force  # we can simply repeat this process for the other planets if we want to find the resultant force  # on the other planets  # what this does is just simply looping through all of the other planets and calculating the force  # they each exert on planet1 then add it all up.  # the beauty lies in the fact that opposite forces just cancel out |

**Finding the instantaneous acceleration of a planet**

|  |
| --- |
| Input a list of planets |
| Input the planet you want the acceleration of |
| Use the above algorithm to find the resultant force on that planet |
| Divide the resultant force by the mass of the planet |
| Return the result |

Link to success criteria:

* The program shall be able to find the acceleration of a planet.

Pseudo code for finding the instantaneous acceleration of a planet:

|  |
| --- |
| planet\_list = input()  planet = input()  resultant\_force = find\_resultant\_force(planet)  acceleration = resultant\_force / planet.mass  return acceleration |

**Finding the velocity at half a step away**

|  |
| --- |
| Instantiate planets with x, y coordinates as well as velocity |
| Input the list of all the planets |
| Input the planet you are considering |
| Input a time difference between time steps (e.g. 1 second per time step) |
| Use the above algorithm to find the current instantaneous acceleration of the planet |
| Use leapfrog integration to find the velocity of the planet at half a time step away by adding (the current instantaneous acceleration multiplied by the difference between time steps) to the (velocity half a time step before) |
| Assume that the velocity at half a step before is the same as the current velocity if the simulation has just begun |
| Store the new velocity as an attribute of the planet |
| Return the new velocity (optional) |

Link to success criteria:

* The program shall be able to find the velocity of a planet at half a step away.

Pseudo code for finding the velocity at half a step away:

|  |
| --- |
| planet1 = planet( x=10, y=5, velocity=(20,4))  planet2 = planet( x=105, y=37, velocity=(23,567))  …  planet\_list = [planet1, planet2, …]  time\_difference = 0.1 # this is in seconds  # let the planet we are considering be planet1  acceleration = find\_acceleration(planet1, planet\_list) # this finds the instantaneous acceleration of   # planet1  # let velocity be called v  planet1.v\_previous = planet1.velocity # assume v at half a step before is the same as current one   # since we have just begun the simulation  planet1.v\_previous = planet1.v\_previous + acceleration \* time\_difference  return planet1.v\_previous  # we can just repeat this for the other planets if we want to consider a different planet |

**Updating the position of planets:**

|  |
| --- |
| Input the list of planets |
| Input the planet you are considering |
| Input the time difference |
| Find the velocity at half a step away by using the algorithm above |
| Calculate the new position by adding the velocity at half a step away multiplied by the time difference to the current position |
| Update the current position |
| Return the current position |

Link to success criteria:

* The program shall be able to update the position of planets.

Pseudo code for updating the position of planets:

|  |
| --- |
| planet\_list = [planet1,planet2,planet3…]  # let the planet we are considering be planet1  time\_difference = 0.1 # this is in seconds  planet1.acceleration = find\_acceleration(planet1)  planet1.v\_previous = find\_v\_previous(planet1, planet1.acceleration)  planet1.position = planet1.position + planet1.v\_previous \* time\_difference  return planet1.position |

Summary:

|  |
| --- |
| This completes the algorithms I need for the simulation. I just need to apply the algorithms I have developed here based on the mathematical models I have referenced to update the position of planets.  Now that I have tackled the problem of the actual simulation, I will move on to tackle the settings. |

Justification of how they form a complete solution:

|  |
| --- |
| All of the algorithms need the ones predefined before them.  They combine to allow for me to get the new position of planets at one time-step away.  Therefore they combine together to form a complete solution for the simulation. |

To tackle the settings I will use the following algorithms:

|  |
| --- |
| Detecting if the user clicks on a button and add a planet at the origin if clicked |
| If the user clicks on a planet then open up a menu to change the settings for the planet |
| Update the planet’s attributes once the user presses enter on any of the field |
| Close the menu if the user presses the cross to close it |

**Detecting if user clicks on a button, then add a planet if clicked:**

|  |
| --- |
| Create a new button. |
| Create a function to add a planet. |
| Link the function to the buton |

Pseudo code for detecting if clicked and adding a new planet:

|  |
| --- |
| button = button()  function clicked():  planet = planet(“untitled planet”, mass=0, position=(0,0), velocity=(0,0), acceleration=(0,0))  # this instantiates a new planet object at the origin  button.clicked\_function = clicked  # this allocates the function to the button object. It automatically detects if it has been clicked |

**If the user clicks on a planet then open up a menu to change the settings for the planet**

|  |
| --- |
| Define a new planet with default attributes |
| Add a button to where the planet is. |
| Create a function to open up a menu |
| Link the function to the planet’s button |
| Have text input boxes in the menu |
| When the user presses enter change the corresponding attribute of the planet |

Pseudo code for opening up a menu if the planet is clicked:

|  |
| --- |
| new\_planet = planet(“untitled planet”, mass=0, position=(0,0), velocity=(0,0), acceleration=(0,0))  new\_button = button(position=(0,0), parent=new\_planet)  # this creates a new planet and maps the button to it   menu = menu()  # create a new menu object – to be implemented later  function open\_menu():  menu.open()  new\_button.clicked\_function = open\_menu |

**Update the planet’s attributes when the user presses enter:**

|  |
| --- |
| Have a menu with text fields |
| Once the user presses on a text field start recording what they input |
| Check if the user input contains anything that is not allowed |
| Update the attribute corresponding to the planet |
| Wait until the user presses the x to close the menu |

Pseudo code for updating the planet’s attributes when the user presses enter:

|  |
| --- |
| planet = planet() # initialise a planet with default attributes  button = button(parent=planet)  menu = menu(parent=button)  function open\_menu(menu):  menu.open()  while True:  if button.pressed = True: # if the planet has been pressed then open up a menu  open\_menu(menu)  if menu.close.pressed = True: # if the button to close the menu has been pressed  menu.close()  for text\_field in menu.text\_fields: # iterate over all the text fields in the menu  if text\_field.clicked:  text\_field.capture() # collect user input   text\_field.data.sanitise() # make sure the user input is allowed  update(text\_field.attribute, text\_field.data) # update the planet’s attribute with the input |

Summary:

|  |
| --- |
| This completes the algorithms needed for setting the attributes of the planet when the user presses on a planet. |

Justification of how it forms a complete solution:

|  |
| --- |
| The user needs a way of changing/interacting with the planets to change the mass and gravitational constant. The menu provides a way of changing the mass of the planets and the velocity and acceleration. The algorithms link together to provide that menu. |

### Testing These Algorithms

The algorithms I will use are similar to normal programming as they use floats/decimals that can be tested using trace tables. I will explain the test data for each algorithm to be used during development.

During development I would most likely use an Excel spreadsheet as using tables in Word is highly inefficient.

Vector subtraction:

|  |  |  |  |
| --- | --- | --- | --- |
| Vector 1 | Vector 2 | Expected | Result |
| (0,1) | (1,0) | (1,-1) |  |
| (1,1) | (1,1) | (0,0) |  |
| (“hello”, “world”) | (“should”, ”error”) | Erroneous data, produces an error |  |
| (123,594) | (34,10) | (89,584) |  |
| (-1000,-210) | (50,50) | (1050, 260) |  |
| (1.1,3.4) | (20.1,10.2) | (19,6.8) |  |

Finding the distance between two vectors:

|  |  |  |  |
| --- | --- | --- | --- |
| Vector 1 | Vector 2 | Expected | Result |
| (0,0) | (0,0) | 0 |  |
| (100,50) | (123,50) | 23 |  |
| (“f”, “g”) | (“he”, “No”) | Error |  |
| (30,40) | (40,50) | 10 \* sqrt(2) |  |

Dividing a vector by a scalar:

|  |  |  |  |
| --- | --- | --- | --- |
| Vector | Scalar | Expected | Result |
| (10,5) | 5 | (2,1) |  |
| (46,18) | 2 | (23,9) |  |
| (57,9) | 3 | (29,3) |  |
| (5062,10240) | “e” | Error |  |
| (1231,50) | 0 | Divide by zero error |  |

Finding the unit vector between two points:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vector 1 | Vector 2 | Expected vector from subtraction | Expected | Result |
| (10,4) | (20,3) | (10,-1) | (0.995,-0.0995) |  |
| (3,4) | (4,3) | (1,-1) | (0.707,-0.707) |  |
| (0,0) | (0,0) | (0,0) | (0,0) |  |
| (“hello world”,) | (“wrong data”) | Error | Error |  |
| (3.4,5.2) | (5.6,1.2) | (2.2,-4) | (0.482, -0.876) |  |

Finding the force a planet exerts on another:

As the planets are far too numerous and a test table would have far too many different variables, I have chosen to use a database-based method where I will have a table for the planets and the force the planets exert on one another.

The planet table:

A screenshot of a computer screen

Description automatically generated

The expected results table:

A close up of a sign

Description automatically generated

I have used excel to generate the results using the algorithms as described before.

Finding the resultant force on a planet:

I can get the software to output the resultant force of one planet then compare it with the sum of forces that affect the same planet in the excel sheet.

The GUI part of the solution requires actually interacting with it, so it does not lend itself to test tables. I will test it during development.

## Graphical User Interface

A few components of the GUI that needs to be done. They will be listed below:

|  |
| --- |
| Main window |
| Settings menu |
| Planet settings menu |

The overall GUI needs to be intuitive for the user to use. The GUI should have 3 different things: main window, settings menu, and planet settings menu.

The main window is the place where I would actually show the simulation of the planets.

The settings menu is the place where the user would change the gravitational constant and the accuracy of the model.

The planet settings menu is the place where the user would change the mass, position, velocity and acceleration of the planets.

Main window:

A computer screen with a black screen

Description automatically generated

Links to success criteria:

* Program shall allow for the user to change the mass of the planet.
* Program shall allow for the user to change the gravitational constant.
* The user interface shall be easy and intuitive to use.
* Advanced settings should be tucked into another menu.
* There should be a play button for the user to play the simulation.

Currently, the window is set to “Main” instead of “Settings”.

The “Main” tab is where the simulation would be plotted. This lets the user see the actions/orbits of the planets.

The “Settings” tab is where the user would adjust settings that affect the overall application. The user would change things such as the gravitational constant, choose a different calculation method, and change the time step that is used in the calculation as described in the algorithms section. I will put a section about the “Settings” tab later.

The side menu is used to adjust the parameters of the planets. The user would change the settings such as mass, position, velocity and acceleration when the simulation is not running. Each of the text input fields would be record-locked when the simulation is paused. When the user tries to alter things when the simulation is running the menu would be record-locked and an alert needs to be sent saying something similar to “Cannot alter planet parameters while simulation is running”.

The empty boxes next to each of the parameters in the side menu would display the current value and allow for the user to edit it while the simulation is at its starting stage (e.g. current time of simulation = 0).

The “Play/Pause Sim” button allows for the user to play and pause the simulation. However, even when paused, if the simulation is not at its starting stage, the user should not be able to alter the parameters of the planet, as that breaks the conservation of energy of the planet.

The “Reset Simulation” button allows for the user to bring the simulation back to its original state. What the user has made before the simulation has begun will be restored. For example if I set up the solar system, and press play, but I reset it, it would return to the solar system at the starting point, instead of just a black screen.

Settings Menu

A screenshot of a computer

Description automatically generated

Links to success criteria:

* Advanced settings should be tucked behind another menu.
* The user interface shall be easy and intuitive to use.
* The program shall allow for the user to change the gravitational constant.

This settings menu allows for the user to adjust a couple of things.

First of all, the time step size allows for the user to change how much time passes for each time step.

Ideally, we would have the smallest time step possible, for higher accuracy. However, this is not always possible in a lower-end computer. The user may want to have a lower accuracy in return for a faster application, or a higher accuracy if their laptop can handle it, as Steven has discussed. People are willing to wait for a little bit for simulations, but if it is too slow then the user has the option of changing how accurate the model is.

The gravitational constant part of the GUI allows for the user to adjust the gravitational constant with either a scroll bar or text input. The input needs to be checked before setting it (e.g. cannot be less than zero).

The calculation algorithm part allows for the user to select a different algorithm to use for calculating the position of the planets. I plan to add this once I am done with the main bulk of the program, with it including Euler’s method and Runge-Kuta method if possible, though it is not a main focus of the project.

Diagram showing how the GUI interface works/links:

A diagram of a structure

Description automatically generated with medium confidence

I will need to ensure that each part of the GUI are working as intended. To do this I will use the following checklist to see if each functionality behaves as intended:

|  |  |  |
| --- | --- | --- |
| Functionality | Expected behaviour | Actual behaviour |
| Tab changes | The tab should switch to the settings menu when the settings tab is clicked. | To be tested |
| Sliders | The value on the left hand side of the slider should increase as you drag the slider to the right, and decrease as you drag the slider to the left. | To be tested. |
| Drop down selection | The menu should drop down showing a few things the user can select. Upon clicking on one of them the drop box should close and update the value of the variable linked to the drop down box. | To be tested |
| Other buttons | They should do what they’re intended for as the user click on them | To be tested. |

## Explanation of User Interface Choices

Having two tabs:

* Currently I have two tabs, one is the main window and the other one is the settings window.
* I think that having two tabs makes it easier to abstract away the different information and help categorize them.
* It separates fundamentally different GUI elements which do different things to make it more clear and less cluttered.

Having a side menu:

* We need a way for the user to see the information about a planet.
* While it is possible to create a display above the planets in the simulation window, it involves having the menu move with the planets which is rather complicated.
* For a prototype of the system I have chosen to use a side menu to abstract away movements of the menu.
* The planets may be moving very fast which makes it harder for the user to select parts of the menu to edit.
* Having a side menu which is fixed in position avoids this problem.

Use of scroll bars:

* Scientific numbers can get overwhelmingly big, with mass of planets going up to 10^23 and more, so it is hard for the user to come up with big numbers as we tend to speak of numbers that are orders of magnitudes smaller in daily life.
* To prevent the user from entering a number that wouldn’t make sense (e.g. mass of planet being 1 kg), I can use a scroll bar and set reasonable bounds (which can be adjusted in the settings, a feature I will make once the prototype is complete). This allows for the user to simply increase it without worrying about whether it makes sense or not.

## Usability Features

A few usability features I have considered are:

* Making the buttons labelled.
* Having large, clear text.
* Colour coding the play button to show if the simulation is running.
* Using scroll bar to make it easier to change values.

Links to success criteria:

* The user interface shall be easy and intuitive to use.

Making the buttons labelled:

* I have named the buttons in the pictures, and they describe what the user is actually changing.
* For example, in the side menu, all of the text input fields are next to text showing the user what it is, and which planet it is for.
* The tabs are labelled with “Main” and “Settings” and this should be easy enough to understand.
* Justification: The user needs a way to know what the buttons are for.

Having large, clear text:

* Whilst the pictures may look small on the report, when the user is running the application, it will take up the full screen.
* I can further resolve this by adding a text size setting in the settings tab.
* The user would then be able to adjust the size of the text as needed.
* Justification: The user needs to be able to read the text which shows what the buttons are for.

Colour coding to show if simulation is running:

* The user needs a way of knowing if the simulation is running.
* I can colour code the play button to show if it is running.
* When it is running, I can use green to show it is running.
* When it is paused, I can use grey to show that it is paused.
* This allows the user to see if the simulation is running.
* Justification: The user needs a way to tell if the simulation is paused or if it is frozen.

Using scroll bars to make it easier to change values:

* I plan to implement a scroll bar for the settings menu where the user can adjust the gravitational constant and the time step size.
* This provides guidance as to what values are accepted to the user, instead of relying on the user to input a value that makes sense for the program.
* This relieves the load on the user as they do not have to try many values to find ones that are suitable.
* Justification: The user needs a way of changing the values but having the user type values with something like \*10 ^ 8 makes it hard for the user to come up with values. I can allow for the user to type it as well but I would need to validate. Instead I can set suitable boundaries for the user to mess around with, for them to learn about the physics while abstracting away some of the input procedure.

## Key Variables/Key Data Structures/Key Classes

Despite what I have demonstrated in the algorithm section, I think that I will not be using global variables as this can easily get messy with lots of planets. Instead, I will use an object-oriented approach and have each planet as objects instead. As such, there is no real key variables as I plan to have everything more dynamic. The planet objects will be stored inside of a data structure such as a dictionary.

**Key classes:**

**The vector class:**

The vector class is needed for vector calculations in the algorithms section. I can easily find a library online that does 2d vectors, but I can also implement the vectors itself. As it is not too  
complicated I plan on undertaking building a custom vector class. The benefit of this is that I   
can ensure I know what the methods are doing.

|  |  |  |
| --- | --- | --- |
| Attribute | What it does | Validation |
| X | X coordinate of the vector | Must be a number |
| Y | Y coordinate of the vector | Must be a number |
| Magnitude | Size of the vector | Must be a number |

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Parameter(s) | What it does | Validation |
| add() | vector | Adds another vector to itself | Input must be a vector object |
| subtract() | vector | Subtract another vector from itself | Input must be a vector object |
| multiply() | vector | Find the dot product of itself and another vector | Input must be a vector object |
| divide() (by scalar only) | num | Divide itself by a scalar number | Input must be a number |
| unitVectorTo() | vector | Find the unit vector that brings you from the current vector to the vector provided in the parameter | Input must be a vector object |
| findMagnitude() | self | Returns the magnitude of itself, changing its magnitude attribute to the new value. | No inputs, must return a number. |

**The planet class:**

The planet class is needed for N-body simulation as I will have many different planets in the simulation and it would be very messy if I wanted to create a new global variable for each of the attributes when I add a planet, so it is much simpler to just use a planet class.

|  |  |  |
| --- | --- | --- |
| Attribute | What it does | Validation |
| Position | A vector describing where the planet is at | Must be a vector object |
| Velocity | A vector describing how fast and in what direction it is moving | Must be a vector object |
| Acceleration | A vector describing how much and in what direction the velocity is changing | Must be a vector object |
| Mass | A scalar describing how much stuff the planet has, essentially how “heavy” it is | Must be a number |
| Velocity\_half\_step | Stores the velocity at half a step before that will be used in further calculations | Must be a vector object |

|  |  |  |  |
| --- | --- | --- | --- |
| Methods | Parameters | What it does | Validation |
| info | self | Return a list of all of its own attributes | N/A |
| find\_force | Planet | Return the force exerted on self by the planet input | Planet must be a planet object |
| find\_resultant | self, planetList | Take a list of planets and computate the resultant force all of the other planets exert on it.  Returns a force vector | planetList must be a list of planet objects |
| find\_acceleration | self, planetList | Calls find\_resultant to find the resultant force.  Divides the resultant force by self.mass to find the acceleration.  Returns the acceleration vector. | planetList must be a list of planet objects |
| find\_new\_pos | self, planetList | Calls find\_acceleration to find the acceleration of the planet.  Uses the algorithm as described in the algorithm section to find the new position vector of the planet.  Returns a position vector. | planetList must be a list of planet objects |
| update\_pos | self | Calls find\_new\_pos to find the new position, then update it on the screen. | N/A |

**Key variables:**

|  |  |  |
| --- | --- | --- |
| **Key variable** | **What it does** | **Validation** |
| planet\_list | Stores a list of the planet objects of planets currently in the simulation | Check if items in the list are planet objects 🡪 required for calculating acceleration |
| simulation\_running | A Boolean variable storing the current state of the simulation.  If false 🡪 simulation is paused.  If true 🡪 simulation is running. | Does not really require validation as it is not a user input |
| timestep\_size | The difference in seconds between time steps for the simulation | Check if it is a floating-point number, as it has to be a number, not string or anything. |
| gravitational\_constant | A variable that holds the current value of the gravitational constant used in the simulation. | Check if it is a floating-point number, as it has to be a number, same as above. |
| simulation\_method | A string variable to indicate which simulation method we are using. | Check if string exists in the list of allowed simulation methods |

## Explanation and Justification of This Process

The project’s task is large and complicated. However, if I split it into many different parts the development can be made easier and more modular in nature. I have divided the task into two sections.

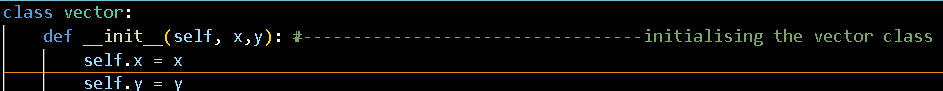
The first section is the algorithms section. It is in this section where I will be tackling the mathematics behind plotting the orbits of planets. Firstly, the system shall consider each of the planets individually. It will then iterate over a given list of planets that exist in the system, finding the position of the two planets. It will then use the two positions to find the distance between the two. Furthermore, it would then computate the vector that takes you from the planet being currently iterated over to the one we are considering. The algorithm then divides the vector we found in the previous step by the distance between the planets to find the unit vector that takes you from the planet we iterate over to the planet we are currently considering. Using Newton’s law of gravitation, we can substitute the value of the gravitational constant set by the user, the masses of the two planets, the distance between them, and the unit vector between them to find the force exerted on the planet we are considering by the planet we are iterating over. After the iteration, we add up all the forces exerted on the planet we are considering to find the resultant force on the planet we are considering. After this, we simply consider the remaining of the planets, repeating the process to find the resultant force for all of the planets. After we have the force, we can iterate over all of the planets and divide the force by their masses to find the acceleration. This step may be put into the for loop before to increase the algorithmic efficiency. Once we have the acceleration of the planet, we can use the velocity at half a step before to find the velocity at half a step away by adding the product of acceleration and time to the velocity at half a step before. To find the new position of the planet, we simply add the product of the velocity at half a step away and the change in time to the current position. This allows us to find the position of the planets at one time step away. We can then use this to simulate the orbit of planets.

The second module is the GUI section. It is in this section where I will handle all the GUI. With the diagram within the GUI section, we can see that the main window utilizes the algorithms to calculate the position of the planets and plot it. The settings in the menu are linked to the calculations and they are essentially knobs and dials we can turn to adjust how the planets are calculated. Breaking down the GUI into different menus (e.g. side menu, settings menu, main window) allows for a more modular development of the system to make it easier to have measurable success and progress.

# Development

## Stage 1:Implementation of the Vector Class

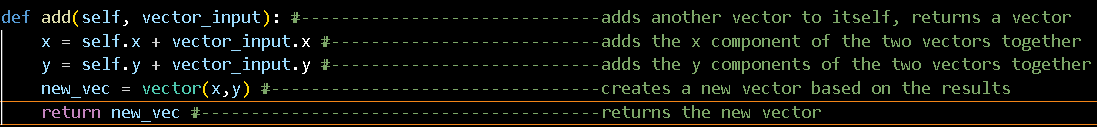
I have initialised the vector class giving it x and y attributes:



It is pretty self-explanatory, so I have not added comments on what the two lines do.

**The Add Method**

The second method I have created is the add method specified in the design.



All it does is that it adds the x and y components of the vectors together and return a new vector. This is required for adding forces together to find resultant force, for example.

Link(s) to success criteria:

* The program shall be able to calculate the orbit of planets.
* The program shall be able to find the resultant force on a planet.

To test the add method I will use the following data:

|  |  |
| --- | --- |
| Vector 1 | Vector 2 |
| (0,5) | (5,1) |
| (3,2) | (5,802) |
| (“e”, “hello”) | (“B”, 9) |
| True | False |

And I will be using the following test code:

A screen shot of a computer program

Description automatically generated

Full test:

|  |  |  |  |
| --- | --- | --- | --- |
| Vector 1 | Vector 2 | Expected | Output |
| (0,5) | (5,1) | (5,6) | (5,6) |
| (3,2) | (5,802) | (8,804) | (8,804) |
| (0.401,8) | (0.23, 0.55) | (0.631, 8.55) | (0.631, 8.55) |
| (“e”, “hello”) | (“B”, 9) | Reject input | Type Error |
| (True, False) | (False, False) | Reject input | (1,0) |

I have forgotten to check for strings when initialising a vector. To do this, I use the following function:

A screen shot of a computer

Description automatically generated

It takes a number, for example the x in our initialisation. It then checks if “-“ or “.” is inside of it and removes it for the check, as it triggers isnumeric() to become false. Then we run a check to see that everything is a number and then accept it. Otherwise we deny it.

Using the following code to do a quick test on the function:

A black screen with colorful text

Description automatically generated

Yields:



So we can tell that the code is working as intended. Now to implement this in the vector class.

A screen shot of a computer

Description automatically generated

Running the testing code:

A screenshot of a computer program

Description automatically generated

Yields:



So we know that the code is preventing us from using unintended characters. Now we just need to apply the check for the addition method. In the future when dealing with the user input I can reuse the check\_position\_valid function to check if the user input is accepted before setting it to prevent any errors, and to ask the user to enter again.

A screen shot of a computer

Description automatically generated

If any of the components are invalid, then we void the entire operation. Returning None allows for the other parts of the application to function without crashing due to the error. We have seen how adding works for normal numbers so I will omit testing for it again. Now it is time to implement subtraction.

**The Subtraction Method**

Though I have explained that we are doing the second vector minus the first vector, I am going to implement self minus the input. So we would have self.x – input.x, etc. To get the same effect, we can just call the subtraction on the other vector, and this way it makes more sense. This is required to find the unit vector and distance between two planets.

Link(s) to success criteria:

* The program shall be able to subtract two vectors.
* The program shall be able to find the distance between two vectors (uses subtraction)

A screen shot of a computer

Description automatically generated

I have modified the check to loop over all of the components instead of one big OR statement to beautify the code for the addition method as well. This takes the same structure as the addition method. I will conduct testing on it now using the following code, and the results are below the code:

A screen shot of a computer code

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Vector 1 | Vector 2 | Expected | Output |
| (10,50) | (5,2) | (5,48) | (5,48) |
| (5,2) | (10,7) | (-5,-5) | (-5,-5) (after bugfix) |
| (-19,69) | (10,8) | (-29,61) | (-29,61) |
| (9.7, 1.2) | (6.5, 0.5) | (3.2, 0.7) | (3.1999999, 0.7) |
| (“hello”, 5) | (2,3) | None | (None,None) post fix |
| (True, False) | (False, True) | None | (None, None) post fix |

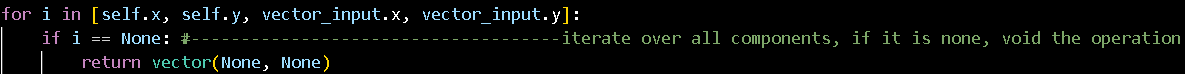
During testing I got a very unexpected bug from the vector class. When using the third test case, it triggered an error saying that list has no property .isnumeric() within check\_position\_valid. I have successfully identified that it was due to me not joining the string back together causing this error in this line:



I have replaced it with this line:



Another bug I got was that the program could not check the vector\_3.x because it is None for test case 5, which is expected. I need to have a way to handle the case where the vector is None. I will simply set the two attributes to None when returning the vector in the addition method and subtraction method.



This way, if I use the values of the vector that is not valid, it also invalidates any operation on the same vector. Test case 5 now yields (None, None) instead of giving an error. The same applies for the last test case. Now we have verified that addition and subtraction method work.

Another very unexpected bug was test case 4 yielding 3.1999999999993 instead of 3.2. It is unclear how this is and I don’t know how to fix it. When using a number with more decimal places it seems to be resolved. Using a new test case with vector 1 = (0.58482, 8.2593929) and vector 2 = (6.55592929, 0.559692), it yields (-5.97110929, 7.6997009) which is precise. For my program I might possibly require test case 4 to have more trailing zeroes to preserve the accuracy, but this is unlikely to cause any issues so I leave it there.

Time to implement the multiplication methods. (Note: I have modified check\_position\_valid and renamed it to isNum to make more sense)

**The Multiplication Methods**

The multiply method requires splitting into two separate functions: a dot product function and a scalar multiply function. A dot product takes two vector input while a scalar multiply takes a vector and multiplies it with a scalar. The results are very different so I have decided to implement separate functions to make it easier to use in the future. I will first implement the dot product function.

Link(s) to success criteria:

* The program shall be able to find the resultant force of planets.
* The program shall be able to plot the orbit of planets.

A screen shot of a computer

Description automatically generated

All the dot product function does is that it:

* Takes two vectors.
* Multiply each component together (e.g. vector1.x multiplies by vector2.x)
* Take the sum of all those products.
* Return the sum.

Note that we are returning None instead of (None, None) in the case of an erroneous input because in normal use we expect the dot product to be a scalar.

Testing with the following code yields the outcome below it:

A screenshot of a computer program

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Vector 1 | Vector 2 | Expected | Output |
| (3,4) | (5,6) | 39 | 39 |
| (-10,3) | (5,-9) | -77 | -77 |
| (0.58482, 8.2593929) | (6.55592929, 0.559692) | 8.456754698 | 8.4567546983646 |
| (“Hello”, “World”) | (“String”, “Input”) | None | None |
| (True, False) | (False, True) | None | None |

Test case 3 returns a number that is not the same as the expected output but that is possibly due to the inexactness of binary floating point. Plus, all the digits from the expected output match up, and the error is quite small, so it is acceptable. Moving on to multiplying with scalar.

Link(s) to success criteria:

* The program shall be able to find the resultant force on a planet.
* The program shall be able to plot the orbit of planets.

A screenshot of a computer

Description automatically generated

All it does is that it:

* Takes a vector and scalar input.
* Multiplies the vector by the scalar.
* Returns the new vector without changing the old vector.

Testing with the following code yields the results below it:

A screen shot of a computer program

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Vector 1 | Scalar input | Expected | Output |
| (5,8) | 7 | (35,56) | (35,56) |
| (-10,-20) | 3 | (-30,-60) | (-30,-60) |
| (5.5,-2.78) | 2 | (11,-5.56) | (11.0,-5.56) |
| “hi” | 9 | None | Error |
| (“hello”, “world”) | 10 | (None, None) | (None, None) |

I got an error running test case 4, but that only affects the program if I fail to force the user to enter two values in a field, and it defaults to blank anyways so this does not affect my project massively. I can safely say that this function works as intended. Moving on to division method.

**The Division Method**

Link(s) to success criteria:

* The program shall be able to plot the orbit of planets.
* The program shall be able to calculate the resultant force exerted on a planet.

A computer screen with text

Description automatically generated

All it does is that it:

* Takes a vector and scalar input.
* Divide the x and y components of the vector by the scalar.
* Returns a new vector with the divided components without changing the original.

Testing with the following code yields the below results:

A screen shot of a computer program

Description automatically generated

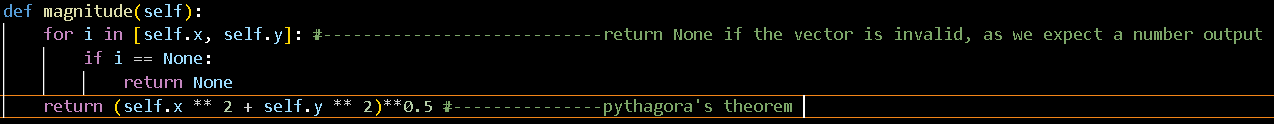
|  |  |  |  |
| --- | --- | --- | --- |
| Vector 1 | Scalar | Expected | Output |
| (5,10) | 4 | (1.25,2.5) | (1.25,2.5) |
| (-50,2) | -5 | (10,-0.4) | (10.0, -0.4) |
| (1.2, 10.8) | 0.4 | (3,27) | (2.999999999,27.0) |
| (“true”, “false”) | “YES” | (None, None) | (None, None) |
| (True, False) | “hello world” | (None, None) | (None, None) |

All test case passed. Can safely move on. As finding the unit vector requires finding the magnitude of a vector it makes more sense to implement finding the magnitude before implementing finding the unit vector.

**The Magnitude Method**

Link(s) to success criteria:

* The program shall be able to find the unit vector between two vectors.
* The program shall be able to plot the orbit of planets.



All it does is that it:

* Checks if vector is valid.
* If so, take the sum of squares of the x and y components.
* Then take the square root of the sum of squares.
* Return the square root.

Testing with the following code and data yields:

A screenshot of a computer program

Description automatically generated

|  |  |  |
| --- | --- | --- |
| Vector | Expected | Output |
| (3,4) | 5 | 5.0 |
| (-5,-12) | 13 | 13.0 |
| (6.301, 9.103) | 11.07100763 | 11.071007632550888 |
| (True, False) | None | None |
| (“test”, “case”) | None | None |

All test case passed (with test case 3’s error being within an acceptable range, since all of the significant places of the expected value match up with the output).

**The Unit Vector Method**

Link(s) to success criteria:

* The program shall be able to find the unit vector between the two planets.
* The program shall be able to plot the orbit of planets.

A computer screen shot of a black background

Description automatically generated

All it does is that it:

* Takes a vector input.
* Finds the vector that takes you from self to the vector input.
* Finds the magnitude of that vector.
* Divide the vector by its magnitude.
* Return the outcome.

Testing with the following code and data yields:

A screen shot of a computer program

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Vector 1 | Vector 2 | Expected | Output |
| (5,19) | (100,6) | (0.9907665955,  -0.1355785868) | (0.9907665955,  -0.1355785868) |
| (-10, 20) | (50,-24) | (0.8064049959,  -0.5913636636) | (0.8064049958557055,  -0.5913636636275174) |
| (6.29, 7.79) | (5.102, 10.50) | (-0.4014924227, 0.9158623447) | (-0.40149242271092894, 0.9158623447362102) |
| (“Test”, “Case”) | (“Return”, “None”) | (None, None) | (None, None) |
| (True, False) | (False, True) | (None, None) | (None, None) |

All test cases have passed.

On a small note, after reviewing with my stakeholders and supervising teacher it has been recommended to me that I should use operator overloading. This would allow me to use the methods I have already created but in a smarter way. For example, to use the addition method if we use operator overloading, it would simply be vector1 + vector2 instead of having to do vector1.add(vector2).

Upon scrutinizing the success criteria, I have realised that I forgot to implement a way to find the distance between vectors directly. I will now implement this.

**The Distance From Method**

Link(s) to success criteria:

* The program shall be able to find the distance between two vectors.

A screenshot of a computer program

Description automatically generated

This uses operator overloading. For example, say I wanted to find the distance between vector1 and vector2, I would simply write vector1 >> vector2 to find it. Testing with the following code and data yields:

A screen shot of a computer code

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Vector 1 | Vector 2 | Expected | Output |
| (0,3) | (4,0) | 5 | 5.0 |
| (5,12) | (-5,-10) | 24.16609195 | 24.166091947189145 |
| (9.50501,100.5592) | (204.60703,332.5060) | 303.0909372 | 303.09093724214256 |
| (“hello”, “world”) | (“test”, “case”) | None | None |
| (True, False) | (False, True) | None | None |

All test case passed, and so the vector class is complete.

I have also added this line to the initialisation of a vector object:



This means that for any vector I create using the class I can use the .size to find its size.

After consulting with my stakeholders and supervising teacher, we have concluded that validation is not necessary for the internal mechanics of the vector class, and is only necessary for when we have user input. This validation can be implemented in the UI part of the development. As such I will be removing all validation in the vector class.

## Review of Stage 1

**What has been completed:**

I have implemented a vector class that allows the program to add, subtract, dot, and find the unit vector of two vectors. It can also multiply and divide vectors by scalar. The class also supports finding the magnitude of a vector. The attributes of the class are x, y, and size, with x corresponding to the component of the vector in the x axis, and y corresponding to the component of the vector in the y axis, and size being the magnitude of the vector.

**How it was tested:**

Each part of the function was tested in a modular way. Separately I have created a test table with values and their expected outcomes. Any unexpected outcomes are rectified with some of them being acceptable due to them resulting from the limitations of binary arithmetic, which I cannot resolve. Other unexpected outcomes are solved through debugging and more sanitisation of the inputs. Only after verifying that all of the functions behave as intended have I moved on.

**Success criteria(s) met:**

The vector class is required for many operations in the future. To find the force

Criteria(s) met are listed here:

* The program shall be able to subtract two vectors.
* The program shall be able to find the distance between two vectors.
* The program shall be able to divide a vector by a scalar.
* The program shall be able to find the unit vector between two vectors.

**Deviation from the initial design:**

Initially I have specified that the vector subtraction algorithm should have:

* Input vector – self

But now I have implemented:

* Self – input vector

Other than that, all of the other algorithms are the same as the design.

**Summary of the overall project at this stage**

Right now, we only have the vector class that will be used for calculating planetary motion. It has been tested using new test data, and I will be using the test data in the design section post-development to further verify that the overall project fulfils the success criteria.

## Stage 2: Implementation of the Planet Class

**Initialisation**

As per the design, we need the following attributes:

* Position
* Velocity
* Acceleration
* Mass
* Velocity\_half\_step

A screen shot of a computer

Description automatically generated

As discussed before, we do not need to perform validation in the class as we only need to do validation when we have user input.

**Finding the force exerted on it by another planet:**

Link(s) to success criteria:

* The program shall be able to calculate the force a planet exerts on another.
* The program shall be able to calculate the resultant force experienced by a planet (this will be a part of the function for that)

A computer screen with green text

Description automatically generated

All the code does is that it:

* Takes the masses of the pair of planets.
* Takes their positions.
* Calculate the distance between them.
* Calculate the unit vector from self to the input.
* Use Newton’s law of gravitation (I have moved the negative in front to the back, and the unit vector is the opposite of what I’ve said in the design, but this is just the negative version on it, so clearly the formula is the same)
* Return the force.

Running the following code with the following test data yields:

Code (will add all planets in the test data during testing):

A black screen with colorful text

Description automatically generated

A screenshot of a computer program

Description automatically generated

All the code does is that it takes all of the test data (planets and expected force output) and compare it to what you would get using the code to propagate the forces. Then it compares it to the test data which we use, and if the percentage error is less than 0.01% we say that it is acceptable.

Note: The find\_pairs function is not a part of the design but I need this function in order to find all the pairs of planets to calculate the force for. This was tested but it is not significant enough to warrant a whole section. All it does is that it loops through an array, and make a list of all the items that come after it and create pairs for each of them. E.g. if I have a list [1,2,3,4,5,6,7,8,9…] then it starts with 1, makes a list of [2,3,4,5,6,7,8,9…] and then create pairs [1,2], [1,3], [1,4], [1,5] etc. Once done, the item 1 is ignored because all possible pairings with 1 have already been marked so we only need to do the rest. This is repeated until all possible pairings are marked.

Data:

A blue background with black numbers

Description automatically generated

Expected output (Rows N-Q):

A screenshot of a computer

Description automatically generated

Actual outputs:

A computer screen with numbers and symbols

Description automatically generated

Clearly all of the tests have passed. Comparing with the actual expected value it is basically the same, with extra digits behind the least significant digit from the expected value most likely resulting from the inaccuracy in binary arithmetic because we cannot represent exact values in binary.

**Finding the resultant force on a planet**

Link(s) to success criteria:

* The program shall be able to find the resultant force exerted on a planet.

It makes more sense to loop over all the planet pairs and then find one of the forces using the find\_force method then multiply it by -1 to find the force on the other planet. This reduces the size of the input by a half and so my code is more optimised. Hence, I will define finding resultant force outside of the class instead.

Testing with the following code:

A screenshot of a computer program

Description automatically generated

All the code does is that it:

* Takes the test data.
* Generates the resultant forces for all planets.
* Compares the resultant forces generated with the expected value from the excel sheet.
* If the percentage error is less than 0.01% then we say it is acceptable.

Actual outputs:

A black screen with numbers

Description automatically generated

Comparing with the test data shows that they are actually basically the same numbers. Clearly all the tests have passed.

**Finding the acceleration**

Link(s) to success criteria:

* The program shall be able to calculate the acceleration of a planet.



All it does is use F = ma and rearrange it to a = F/m and then use the resultant force to find the acceleration. Note that this requires calling find\_resultant\_force before this can be used each time step.